VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Klient nativní XML databáze
Client of a Native XML Database

2009 Pavel Mitko

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Rád bych na tomto místě poděkoval doc. Ing. Michalu Krátkému, Ph.D. za veškerou pomoc a za trpělivost projevenou při vedení mé bakalářské práce. Především bych mu rád poděkoval za spoustu informací a za rady, které mě vždy navedly tím správným směrem.

Abstrakt

Cílem této diplomové práce je implementace klientského softwaru pro připojení a získání požadovaných dat z existující nativní XML databáze. V první fázi dokumentu bude popsán jazyk XML, jeho funkcionalita a také seznámí čtenáře s problematikou dotazovacích jazyků určených pro vyhledávání dat v XML dokumentech. Jedna kapitola je také věnována teoretickému popisu nativních XML databází. Poté již následuje podrobný popis konkrétní zvolené nativní XML databáze, jejího API a vývoje vlastní klietské aplikace, která s touto databázi bude komunikovat. V průběhu práce byla vyvíjena experimentální metoda komprese XML dat a vlastní komponenta sloužící k zobrazování XML dokumentů, jenž byly do aplikace zakomponovány.

Klíčová slova: XML, XPath, XQuery, parser, nativní XML databáze, klient, eXist, komprese, komponenta

Abstract

Target of this thesis is to implement a client-side software for creating a connection to an existing native XML database and obtaining required data from the database. First part of this thesis will describe the XML language, its functionality and also will make viewers familiar with query languages for selecting data withing XML documents. One chapter is also devoted to theoretic definition of native XML databases. Then here comes a detailed exploration of a particular chosen XML database, its API and the development of a client application which will communicate with the database. During this work, an experimental method of XML data compression and own component for representing XML documents were developed. Both those were then integrated into the application.

Keywords: XML, XPath, XQuery, parser, native XML database, client, eXist, compression, component

Seznam použitých zkratek a symbolů

XML	–	eXtensible Markup Language
DOM	–	Document Object Model
LAN	–	Local Area Network
API	–	Application Programming Interface
SGML	–	Standard Generalized Markup Language
W3C	–	World Wide Web Consortium
UTF-8	–	8-bit UCS Transformation Format
UCS	–	Universal Character Set
HTML	–	HyperText Markup Language
PDF	–	Portable Document Format
HTTP	–	HyperText Transfer Protocol
WWW	–	World Wide Web
CSS	–	Cascading Style Sheets
XSL	–	Extensible Stylesheet Language
XHTML	–	eXtensible HyperText Markup Language
DTD	–	Document Type Definition
PCDATA	–	Parsed Character Data
CDATA	–	Character Data
XSD	–	XML Schema Definition
TCP	–	Transmission Control Protocol
URI	–	Uniform Resource Identifier
JDK	–	Java Development Kit
GUI	–	Graphical User Interface
REST	–	Representational State Transfer
SOAP	–	Simple Object Access Protocol
XML-RPC	–	XML Remote Procedure Call
RTF	–	Rich Text Format

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Technologie XML v současné době stále více proniká do nejrůznějších odvětví lidské činnosti. Největší zastoupení má pochopitelně v oblasti informačních technologií, ale najdeme ji např. i v kontextu geografických věd, archivnictví a mnoha dalších. Rostoucí využívání formátů XML však s sebou přináší náročnější požadavky na ukládání a správu takových dat.

Jakožto i u jiných formátů dat, i u XML jsou k dispozici databázové systémy schopné tato data pojmout a spolehlivě uchovávat. Pro XML dokumenty existuje speciální typ databází. Jedná se o tzv. nativní XML databáze. Jejich nejsilnější výhodou je ukládání XML dokumentů v jejich přirozené (nativní) formě.

Ačkoli je dnes těchto XML databázových produktů na trhu celá řada, většina z nich předpokládá vývoj vlastního softwaru, který na základě nějakého definovaného rozhraní API bude k databázi přistupovat a s daty pracovat. Vzhledem k tomu, že nativní XML databáze je pojem relativně mladý a celá tato problematika je neustále ve stádiu výzkumu a vývoje, je takových klientských aplikací nedostatek. Jak je složité a praktické takovou aplikaci vytvářet, se pokusím objasnit v této práci.

Vlastní obsah práce je rozdělen do dvou hlavních částí. První část (kapitoly 1 - 4) seznámí čtenáře s teorií jazyka XML, s oblastí jazyků pro dotazovaní nad XML daty a nakonec s problematikou nativních XML databází. V této části také bude zvolena jedna konkrétní XML databáze, ke které poté bude vyvíjena klientská aplikace.

V druhé části (kapitola 5) již bude popsán vývoj samotné aplikace. Celý vývoj je rozdělen do tří etap: návrh implementace, implementace a testování s vyhodnocením výsledků. Jak již bylo řečeno, oblast nativních XML databází je v neustálém vývoji, hledají se nové techniky, přístupy k datům apod. Proto i v mé aplikaci budou zahrnuty vlastní techniky práce s XML daty - komprese a zobrazování XML dat. Každá tato zkoumaná technika bude podrobně popsána jak v rámci návrhu implementace, tak v samotné implementaci.

2 Technologie XML

Tato kapitola popisuje technologii XML, její vznik, princip, základní rysy jazyka XML a nejčastější použití.

2.1 Popis

Extensible Markup Language (dále jen XML) [11] je značkovací jazyk otevřeného standardu (open standard) pro dokumenty obsahující strukturovaná data. Hlavním úkolem tohoto jazyka je usnadnit informačním systémům šíření a výměnu oněch strukturovaných dat, především pak přes lokální počítačové sítě LAN nebo internet, ale i jinou formou. Vyvinul se postupně z univerzálního jazyka SGML (Standard Generalized Markup Language) a náleží pod správu W3C konsorcia (World Wide Web Consortium), které také XML v roce 1998 standardizovalo. Díky své jednoduchosti a bezplatnému používání se stává čím dál více populárním u široké veřejnosti uživatelů. Slovo Extensible v názvu jazyka můžeme do češtiny přeložit jako *rozšiřitelný* a toto označení nebylo zvoleno jen tak. Je to především z toho důvodu, že si každý uživatel vytváří značky (tagy) sám podle vlastního uvážení. Neexistuje žádná předem daná množina předdefinovaných tagů, záleží pouze a jenom na konkrétním uživateli jaké značky si pro svůj XML dokument zavede. Vzhledem k tomu, jak rychle a s jakou razancí technologie XML proniká do různých oborů informačních technologií, má v budoucnu zajisté velký potenciál. Vždyť i sám Bill Gates prohlásil, že XML je "technologií budoucnosti".

2.2 Hlavní rysy jazyka XML

Výčet bezpochyby nejdůležitějších vlastností jazyka XML:

- bezplatné šíření a užívání (fee-free)
- velmi dobře zdokumentován - specifikace, doporučení (`http://www.w3.org/XML/`)
- nezávislost na použitém softwaru, operačním systému a prostředí
- mezinárodní podpora (používání Unicode, UTF-8 aj.)
- přesně daná syntaxe jazyka, která předchází nekompatibilitě
- vysoký informační obsah (vlastní tagy k hlubšímu označení významu části textu)
- snadná konverze na jiné formáty (MS Word, MS Excel, PDF, HTML apod.)

2.3 Syntaxe XML dokumentu

Každý XML dokument musí splňovat několik kritérií, aby mohl být považován za správně strukturovaný (*well-formed*) a vhodný pro další zpracování. Protože je efektivita jazyka závislá na struktuře a integritě dat, jsou tyto předpisy striktní. O tom, zda je dokument splňuje nebo ne, rozhoduje tzv. syntaktický analyzátor (*parser*).

2.3.1 Pravidla XML dokumentu

- každý XML dokument musí mít daný typ kódování textu (Unicode, v ČR často používaný UTF-8, povolené jsou i jiná kódování)
- dokument má právě jeden root (kořenový) element
- každý neprázdný element je regulérně ohraničen startovací a ukončovací značkou
- prázdný element je ohraničen startovací značkou a ukončen značkou *prázdný element*
- elementy mohou být vnořeny do sebe, nesmí však dojít k jejich překrývání, tzn. každý element (kromě kořenového), pokud je vnořený do jiného elementu, tak pouze kompletně
- v XML dokumentu se u jmen elementů rozlišují malá a velká písmena, např. značka *<Name>* a *<name/>* není validní pár značek ohraničující element *Name*
- prvek použitý s otazníky *<?name?>* se při zpracovávání přeskakuje, používá se mimo jiné při deklaraci XML dokumentu
- poznámky uživatele v dokumentu se značí *<!−−nějaký_komentář−−>*
- pro speciální znaky používané v syntaxi jazyka je třeba použít zástupné řetězce:

- **&**	- pro znak &
- **<**	- pro znak <
- **>**	- pro znak >
- **'**	- pro znak ’
- **"**	- pro znak ”

2.3.2 Ukázka XML dokumentu

Výpis 1: Syntakticky správný XML dokument

```
<?xml version="1.0" encoding="utf−8"?>
<studium>
   <skola name="VSB−TUO" rok="2008">
      <student>Pavel Mitko</student>
      <student>Jan Novy</student>
      <student>Michal Polak</student>
   </skola>
   <skola name="UTB" rok="2007">
      <student>Jiri Vojta</student>
      <student>Petr Maly</student>
   </skola>
</studium>
```

2.4 Sémantika XML dokumentu

U XML dokumentu však často potřebujeme kontrolovat také jeho obsah po stránce sémantické. K tomu, jak už napovídá název, nám slouží analyzátory sémantické. K takové kontrole je třeba vytvořit předpis, který bude podobu správného dokumentu zachycovat. Můžeme do něj zanést jaké elementy v dokumenty mohou, musí, nebo naopak nesmí být obsaženy, v jakém sledu se budou vyskytovat, popř. jaké obsahují atributy atd. Pro vytváření takových předpisů nám slouží dva jazyky. Prvním jazykem je DTD (z angl. Document Type Definition), neboli *definice typu dokumentu*. Více informací o tomto jazyku a ukázku použití v kapitole 2.4.1. Druhým jazykem v pořadí je XML Schema, popsaný v kapitole 2.4.2.

2.4.1 DTD

DTD [4] je ve skutečnosti deklarativní popis XML dokumentu, pomocí kterého jsou následovně sémantické analyzátory schopné ověřit validitu. V praxi se setkáme s už předdefinovanými schématy pro celé skupiny dokumentů se stejným či podobným zaměřením. Příkladem můžeme uvést schémata pro XML dokumentaci knih, novinových článků apod. (především využíváno v evidencích knihoven). DTD předpis je obvykle uložen v externím souboru, není však vyloučeno ani jako součást dokumentu. Při zápisu DTD jsou nám k dispozici také tyto zástupné symboly vyjadřující povinnost výskytu:

+ výskyt alespoň jednou - <1,nekonečno)
? nepovinný výskyt - <0,1>
***** libovolný počet výskytů - <0,nekonečno)

Pro předchozí XML dokument (výpis 1) bychom DTD mohli definovat například takto:

Výpis 2: Ukázka jazyka DTD

```
<!DOCTYPE studium [
  <!ELEMENT studium (skola+)>
  <!ELEMENT škola (student*)>
  <!ELEMENT student (#PCDATA)>
  <!ATTLIST skola
    name CDATA
    rok ( 2006 | 2007 | 2008 ) '2008'>
]>
```

Takový zápis pravidel nám říká, že prvek *studium* musí obsahovat aspoň jeden vnořený prvek s názvem *skola*. Ten dále může obsahovat libovolný počet prvků *student*, který však už neobsahuje vnořené prvky, pouze textovou hodnotu (označení #PCDATA, z anglického Parsed Character Data).

Pro každý takto nadeklarovaný element našeho dokumentu můžeme určit libovolný počet atributů. Atribut je ve skutečnosti informace, která nějakým způsobem přesněji definuje prvek. Názvy atributů opět záleží pouze na nás.

Definice atributů ve výpisu 2 začíná klíčovým slovem ATTLIST. Takto bychom pro prvek *skola* zavedli dva atributy - *name*, který bude obsahovat nějaký text (CDATA z anglického Character Data) a *rok*, jehož přípustné hodnoty jsou dány výčtem tří možností, přičemž standardně nabývá hodnoty *2008*.

Jazyk DTD s sebou nese několik nevýhod. Především to, že v něm nejsou definovány datové typy. Neumožňuje kontrolu přesného počtu výskytů. A v neposlední řadě i fakt, že samotný zápis DTD není ve formátu XML dokumentu.

2.4.2 XML Schema

Dalším jazykem pro popis obsahu XML dokumentu je XML Schema [7], jenž také spadá pod specifikace W3C konsorcia. Správně napsané schéma tímto jazykem je zároveň správně strukturovaný XML dokument, který používá speciálních elementů. Všechny tyto elementy musí patřit do jmenného prostoru *http://www.w3.org/2001/XMLSchema*. Pro tento jmenný prostor se obvykle používá v zápise prefix xs nebo xsd.

Velkou výhodou jazyka XML Schema oproti DTD je přítomnost datových typů [9]. Je také možné definovat vlastní datové typy, nejčastěji restrikcí nebo rozšířením již existujících datových typů. Taková možnost kontroly z XML Schema činí silnější nástroj pro předpis XML dokumentu než je DTD, avšak často hůře čitelným a srozumitelným pro člověka.

Názorná ukázka je vždy nejvýstižnější, proto si uvedeme příklad jazyka XML Schéma a na něm si poté vysvětlíme jeho hlavní rysy. Jde opět o aplikaci na XML dokument uvedený v kapitole 2.3.2:

Výpis 3: Ukázka jazyka XML Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="studium">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="skola" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="student" type="xsd:string" maxOccurs="unbounded" />
                  </xsd:sequence>
                  <xsd:attribute name="name" type="xsd:string" />
                  <xsd:attribute name="rok" type="xsd:int" />
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Celé schéma musí být vnořeno do kořenového elementu s názvem *schema* (*xsd:schema* včetně prefixu). Vzpomeňme, že každý správně strukturovaný XML dokument má právě jeden kořenový element. Nejinak tomu musí být i v případě dokumentu obsahujícím XML Schema.

Pro každý element určujeme jeho typ. K dispozici máme dva typy - jednoduché a komplexní. Mezi jednoduché patří např. řetězec, číslo, datum apod. Pokud však element obsahuje vnořené elementy nebo atributy, musíme použít komplexní typ (ve výpisu *xsd:complexType*). Dále pomocí značky *xsd:sequence* určíme pořadí vnořených elementů vzhledem k nadřazenému elementu. V našem případě prvek *studium* obsahuje pouze vnořený element *skola*, avšak s neomezeným počtem výskytů (vlastnost *maxOccurs="unbounded"*). Element *škola* je dále tvořen vnořeným elementem *student*, který obsahuje hodnotu typu řetězec (*type="xsd:string"*). U tohoto elementu také definujeme dva atributy s názvy *name* a *rok* včetně typů jejich hodnot.

Na obrázku 1 je uvedena hierarchie datových typů obsažených v jazyku XML Schema.

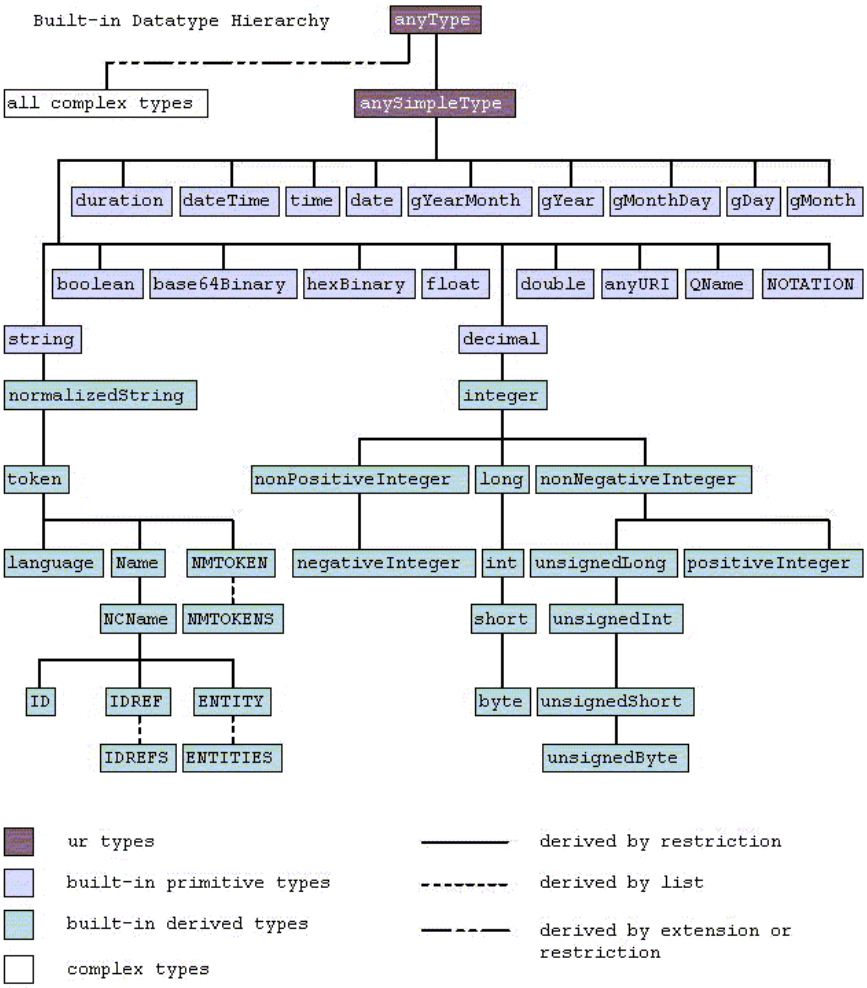
2.5 DOM

Z předchozích kapitol již víme, co je dokument XML a co je schéma popisující jeho strukturu. Nyní se blíže podíváme na to, jak s vlastními daty pracovat.

Objektový model dokumentu (z anglického DOM neboli Document Object Model) [1] zpřístupňuje dokument XML jako stromovou strukturu v paměti a nabízí tak každému programátorovi snadno použitelné prostředí. DOM definuje standardní sadu objektů a rozhraní pro manipulaci s XML, zajišťuje přístup k dokumentům, elementům i atributům. Pomocí rozhraní DOM můžeme vyjádřit celý dokument XML jako objekt, takže je možné pracovat s ním jako s kterýmkoli jiným objektem v systému - k dispozici je kvalitně zdokumentované API (rozhraní pro programování aplikací) s užitečnými metodami a vlastnostmi. Stromovou strukturu XML dokumentu tak, jak ji chápeme při použití objektového modelu dokumentu, ilustruje obrázek 2.

Na DOM se můžeme dívat jako na dynamický objekt s určitou hierarchií. Je důležité si uvědomit, že počítač vidí objekt, kterému my říkáme XML dokument, jako posloupnost bajtů. Protože tato kolekce bajtů má formu prostého textu, může být snadno načten a také přenášen po síti či přes internet.

Aby však počítač mohl pracovat s dokumentem XML a manipulovat s údaji, které jsou v něm uloženy, musí se nejprve dokument změnit v objekt umístěný v operační paměti. V tomto stavu je už lépe přizpůsoben ke zpracovávání různými aplikacemi napsanými v moderních programovacích jazycích. Tato změna se provádí vytvořením instance DOM, která spustí XML analyzátor (parser), a ten rozdělí celý dokument na jednotlivé části. Analyzátor načítá dokument znak po znaku a přitom určuje, zda nově načtený znak patří ke značkování nebo obsahu. V tomto stádiu se také parser dívá, zda k dokumentu náleží nějaké XML schéma nebo DTD, jenž by předepisovaly strukturu dokumentu. V případě, že takové schéma existuje, celé ho načte a poté prověří, zda dokument vyhovuje struktuře, která je v něm popsána. Pokud však analyzátor žádné schéma nenalezne, postupuje podle obecných pravidel správně strukturovaného dokumentu.

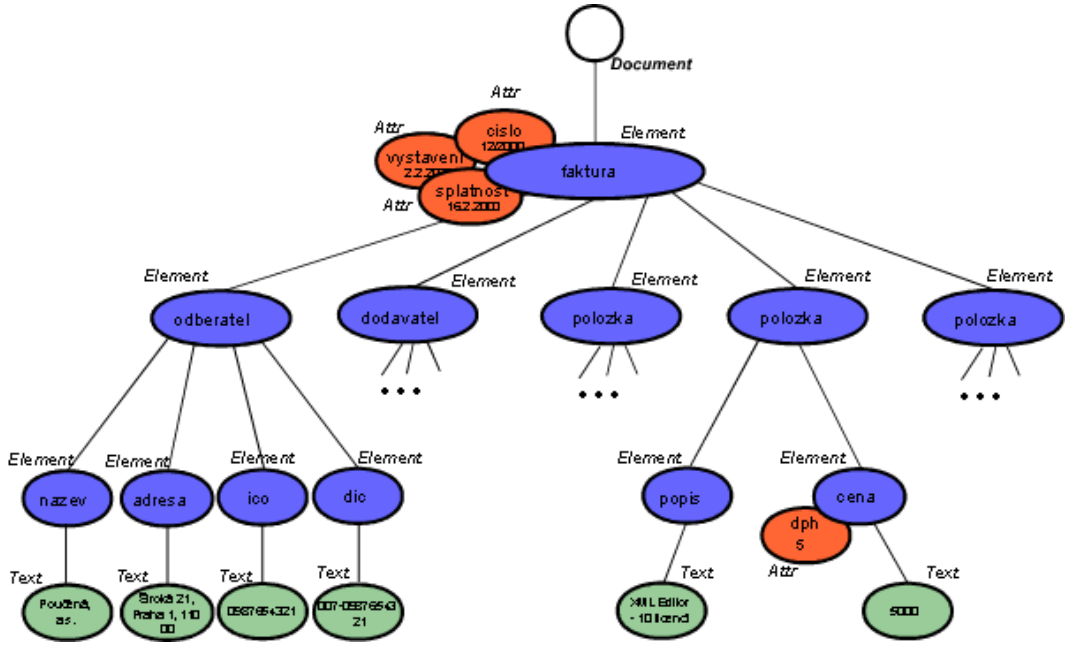


Obrázek 1: Hierarchie datových typů obsažených v jazyku XML Schema, podle [5]

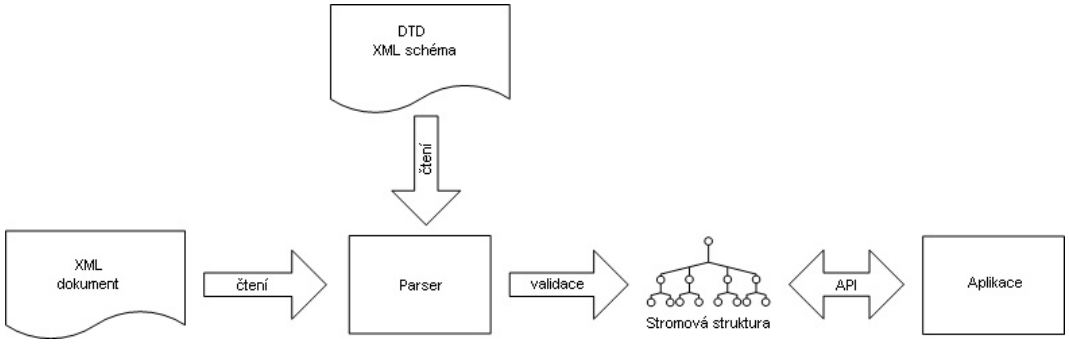
Jakmile je analyzátor s prověřením platnosti hotov a v dokumentu nebyly nalezeny žádné chyby, vytvoří sadu uzlů, které mají určité vlastnosti. V tabulce 1 je uveden seznam možných typů uzlů, které jsou obsaženy v implementaci DOM od firmy Microsoft. Průběh celého procesu vytváření struktury DOM je vyobrazen na obrázku 3.

2.6 Srovnání s jazykem HTML

Stejně jako XML, i jazyk HTML (HyperText Markup Language), který se do podvědomí lidí dostal především díky obrovskému rozmachu internetových stránek (www stránky) za posledních několik let, vychází z původního SGML. Z toho také plyne jejich podobnost v mnoha směrech (především v oblasti syntaxe). Podstatný rozdíl je ale ten, že v jazyku HTML existuje konečná množina značek, které lze v rámci jazyka používat. To ovšem u jazyka XML neplatí. V jazyku HTML jednotlivé tagy prakticky určují vzhled



Obrázek 2: Objektový model dokumentu - DOM



Obrázek 3: Vytvoření stromové struktury DOM

webové stránky - interpretaci, jakou budou data podána uživateli. Kdežto u jazyka XML se vzhled a to, jak bude dokument zobrazen, definuje použitím tzv. stylů. Nejznámějšími zástupci stylových jazyků používaných spolu s XML jsou zcela určitě kaskádové styly (CSS) a XSL (eXtensible Stylesheet Language).

Společností W3C byl z jazyka XML vyvinut nový značkovací jazyk určený pro tvorbu hypertextových dokumentů v prostředí www stránek. Jazyk nese označení XHTML (Extensible HyperText Markup Language) a hovoří se o něm dokonce jako o nevyhnutelném nástupci jazyka HTML. V současné době se intenzivně pracuje jak na nové verzi jazyka HTML 5, tak na XHTML 2.0. U jazyka HTML je naprostá většina softwarových internetových prohlížečů ochotná a schopná tolerovat menší chyby v syntaxi dokumentů, jako

Tabulka 1: Typy uzlů definované v DOM od firmy Microsoft, podle [1]

Název	**Popis**
NODE_ELEMENT	Uzel představující element
NODE_ATTRIBUTE	Uzel představující atribut elementu
NODE_TEXT	Uzel představující textový obsah tagu
NODE_CDATA_SECTION	Uzel představující sekci CDATA ve zdroji XML
NODE_ENTITY_REFERENCE	Uzel představující odkaz na entitu v dokumentu XML
NODE_ENTITY	Uzel představující rozvinutou entitu
NODE_PROCESSING_INSTRUCTION	Uzel představující instrukci pro zpracování z dokumentu XML
NODE_COMMENT	Uzel představující komentář v dokumentu
NODE_DOCUMENT	Tento uzel reprezentuje objekt dokumentu, který coby kořen celého stromu zajišťuje přístup k celému dokumentu XML
NODE_DOCUMENT_TYPE	Uzel představující deklraci typu dokumentu zadanou pomocí tagu <!DOCTYPE>
NODE_DOCUMENT_FRAGMENT	Tento uzel vyjadřuje fragment dokumentu
NODE_DOCUMENT_NOTATION	Uzel představující notaci v deklaraci typu dokumentu

např. opomínání ukončovacích značek elementů, překrývání prvků, používání malých i velkých písmen v názvech prvků. Jak už bylo zmíněno výše, všechny tyto nepřesnosti jsou v syntaxi XML striktně kontrolovány, tudíž ani jazyk XHTML je logicky nepřipouští.

3 Dotazovací jazyky pro XML

Tato část práce nastíní problematiku dotazovacích jazyků určených pro XML. Budou uvedeni někteří zástupci a nejznámější z nich budou prozkoumáni hlouběji v kapitolách 3.2 a 3.3.

3.1 Dostupné dotazovací jazyky

Vzhledem k tomu, že formát XML je schopen data pojmout, je zásadní mít k dispozici prostředky, kterými data opět získáme zpět. Tak jak v reálném životě, i při obnovování dat však chce uživatel často získat jen některé konkrétní informace. Ty, které odpovídají jeho dotazu. Odtud také název *dotazovací jazyky* [8]. Dotazovací jazyk můžeme definovat jako člověku srozumitelný prostředek, kterým je schopen interpretovat své logické dotazy a používat ho pro vyhledávání určitých informací. Každý takový jazyk však má svoje pravidla, která musí dodržovat.

Dotazovacích jazyků pracujících nad XML daty je dnes celá řada, některé už velmi dobře známé, jiné méně. Většina z nich však ve stádiu neustálého vývoje. Mezi nejznámější určitě patří jazyky jako XPath, XQuery, XML-QL (XML Query Language), XQL, X2QL nebo SQL/XML. Jazyku XPath je věnována následující kapitola 3.2 a jazyku XQuery kapitola 3.3.

3.2 Jazyk XPath

Jazyk XPath (zkráceno z anglického XML Path Language) [2] je standardem W3C konsorcia. Při procesu dotazování pomocí tohoto jazyka se pracuje nad stromovou strukturou odpovídající XML dokumentu. K procházení stromu se poté používají tzv. osy (relace mezi uzly stromu), kterými definujeme cestu k námi hledaným datům.

V současné době je již jazyk XPath ve verzi 2.0. Ne každý XML databázový systém podporuje tuto verzi jazyka, ovšem každý by měl podporovanou verzi ve svých specifikacích uvádět. Od verze 1.0 se jazyk liší především přidáním podpory více datových typů [3]. Datové typy jazyka XPath 2.0 jsou uvedené v obrázku 12, který je součástí příloh.

Datovým modelem jazyka je strom, který může být tvořen sedmi druhy uzlů: dokument, element, atribut, text, jmenný prostor, komentář a instrukce ke zpracování. Uzlem *dokument* je myšlen kořenový uzel. Základní částí jazyka je *vyjádření cesty* (path expression). Ta se zapisuje jako posloupnost přechodů mezi uzly oddělených lomítky (podobný zápis jako URL v prostředí internetu nebo adresářová struktura na počítači). Každý přechod je určen třemi složkami [17]:

- osa (axis)
- test (node test)
- predikát (precidate)

Ne všechny tyto položky jsou povinné, nejjednodušší zápis obsahuje pouze část test a má tvar např.:

/A/B/C.
Takový zápis označuje element C, který je potomkem elementu B, ten je potomkem elementu A a ten je zároveň kořenovým elementem celého dokumentu. Výsledkem nemusí být pouze jeden element, v našem případě C, ale může jím být celá množina elementů, které odpovídají této cestě ve stromové struktuře. V zápisu jednotlivých dotazů můžeme, a určitě i hojně využijeme zástupné symboly, které jsou v jazyce XPath k dispozici. Jejich přehled najdeme v tabulce 2.

Tabulka 2: Tabulka symbolů v syntaxi jazyka XPath

Symbol	**Funkce symbolu**
-	jakýkoliv jednotlivý uzel
/	odděluje uzly v cestě
|	sjednocuje uzly
?	0-1 výskyt uzlu
+	1-více výskytů uzlu
*	0-více výskytů uzlu
[]	obsahuje logické podmínky
@	označuje atribut
()	precedence (určení priorit)

U pokročilejších dotazů se již přidává položka osa. Implicitně se používá osa *child* (potomek), ale podle potřeby máme v možnostech jazyka os několik. Zápis os se odděluje dvěmi dvojtečkami. Celý dotaz pak ještě můžeme doplnit o logické podmínky (predikáty), které se uzavírají do hranatých závorek. Jako příklad poslouží např. tento dotaz:
/A/B/child::*[attribute::cena="100"]
který vybere všechny elementy, které jsou potomky elementu B a obsahují atribut cena s hodnotou 100, přičemž element B je potomkem kořenového elementu A.

Os, které definují množinu uzlů relativně k aktuálnímu uzlu, je celkem 13. Uvedeme si je pro lepší pochopení možností dotazování pomocí XPath.

- **ancestor** (předek) - obsahuje všechny předky aktuálního uzlu (uzly ležící blíže ke kořenovému uzlu)
- **ancestor-or-self** (předek včetně sebe) - obsahuje všechny uzly jako ancestor včetně aktuálního uzlu
- **parent** (rodič) - obsahuje rodiče aktuálního uzlu (první uzel blíže ke kořenovému uzlu)
- **child** (potomek) - obsahuje všechny potomky aktuálního uzlu
- **attribute** - obsahuje všechny atributy aktuálního uzlu
- **descendant** - obsahuje všechny uzly, pro které je aktuální uzel předkem
- **descendant-or-self** - obsahuje uzly jako descendant včetně aktuálního uzlu

- **preceding** - obsahuje všechny uzly, které se nachází před aktuálním uzlem, kromě jeho předků
- **preceding-sibling** - obsahuje uzly, které jsou sourozenci aktuálního uzlu a nachází se před ním
- **following** - obsahuje všechny uzly, které se nachází po aktuálním uzlu, kromě jeho potomků
- **following-sibling** - obsahuje uzly, které jsou sourozenci aktuálního uzlu a nachází se po něm
- **namespace** - obsahuje všechny namespace uzly aktuálního uzlu
- **self** - obsahuje aktuální uzel

Několik jednoduchých dotazů pomocí jazyka XPath (aplikované na XML dokument z výpisu 1):

- **/studium/skola/student** - všechny elementy *student*, které jsou potomky elementu *skola* a ten je potomkem kořenového elementu *studium*
- **//skola/student** - všechny elementy *student*, které jsou potomky elementu *skola* a ten je potomkem kořenového elementu
- **//skola[attribute::name='VSB-TUO']** - všechny elementy *skola* s hodnotou VSB-TUO v atributu *name* , které jsou potomky kořenového elementu
- **//skola[@name='VSB-TUO' and @rok** $>$ **2007]/student** - všechny elementy *student*, které jsou potomky elementu *skola* s hodnotou VSB-TUO v atributu *name* a hodnotou větší než 2007 v atributu *rok* a ten je potomkem kořenového elementu *studium*
- **//skola/student[position()=1]** - všechny elementy *student*, které jsou prvními potomky elementu *skola* a ten je potomkem kořenového elementu
- **//skola/student[text()='Pavel Mitko']** - všechny elementy *student* s textovou hodnotou Pavel Mitko, které jsou potomky elementu *skola* a ten je potomkem kořenového elementu

3.3 Jazyk XQuery

Dotazovací jazyk XQuery je stejně jako XPatch produktem W3C konsorcia. Postupně byl vyvinut z jazyka Quilt, jenž vychází z několika dalších jazyků jako XPath 1.0, XML-QL, XQL, SQL. Doporučením W3C se stal 23. ledna 2007. Aktuální verze jazyka je 1.0. V porovníní s XPath se jedná o pokročilejší dotazovací jazyk. Kromě základních výrazů, které přebírá od jazyka XPath (vyjádření cesty, porovnávání, predikáty), poskytuje uživateli

mnohá další vylepšení. Tím jsou nepochybně výrazy typu FLWOR[1], podmíněné výrazy, možnost třídění nebo konstruktory. Výrazy pro určení cesty v XML dokumentu jsou stejné jako v jazyku XPath 2.0.

Pro práci s daty je v XQuery předdefinováno několik vstupních funkcí. Nejdůležitějšími jsou tyto:

- **doc()** - vrací dokument identifikovaný URI
- **collection()** - vrací kolekci dokumentů spojenou s URI
- **root()** - vrací kořen aktuálního dokumentu

Několik jednoduchých dotazů pomocí jazyka XQuery (aplikované na XML dokument z výpisu 1):

- **doc('studium.xml')//skola** - všechny elementy *skola*, který je potomkem kořenového elementu v dokumentu *studium.xml*
- **doc('studium.xml')/studium/skola[student='Pavel Mitko']** - všechny elementy *skola* obsahující element *student* s hodnotou Pavel Mitko, který je potomkem kořenového elementu *studium* v dokumentu *studium.xml*
- **doc('studium.xml')/studium/skola/student[1]** - vždy první element *student*, který je potomkem elementu *skola* a ten je potomkem kořenového elementu *studium* v dokumentu *studium.xml*

Velmi důležitou částí jazyka XQuery jsou tzv. FLWOR výrazy [10], které můžeme přirovnat k SELECT-FROM-WHERE dotazům v jazyku SQL. Název je tvořen prvními písmeny jednotlivých klauzulí tvořící výraz:

- **F** for - asociace jedné nebo více proměnných k výrazu
- **L** let - přiřazení výsledku výrazu proměnné
- **W** where - omezující podmínka
- **O** order by - setřídění
- **R** return - výsledek

```
for $b in doc('studium.xml')/studium/skola
where $b/student = 'Pavel Mitko'
order by $b/@name
return $b
```

Výpis 4: Příklad použití FLWOR výrazu (aplikovaný na XML dokument z výpisu 1)

Příklad uvedený ve výpisu 4 vypíše všechny elementy *skola* seřazené podle hodnoty atributu *name*, ve kterých se vyskytuje *student* s hodnotou Pavel Mitko.

[1] Výslovnost jako "flower".

4 XML databáze

V této části si objasníme co jsou XML databáze, jejich význam v dnešní době a způsob přístupu do nich. Postupně se zaměříme na nativní XML databáze.

4.1 Význam XML databází

Jednou z hlavních oblastí využití XML dokumentů je dlouhodobé uchovávání dat v přehledné formě. To je také principem celé XML technologie. Totiž dokumenty v ní vytvořené jsou svým způsobem samopopisné. To je dáno významem jednotlivých značek, atributů, i samotné struktury dokumentu. Je zřejmé, že dnes vytvořený XML dokument archivující např. knihu ve vědecké knihovně, který je psán podle mezinárodně daných schémat pro tento obor, bude i za několik let jednoduše čitelný a bude plnit svou funkci. Právě toto čím dál častěji vede instituce i jednotlivce zvolit XML jako způsob archivace dat. Potřeba uchovávání velkého množství XML dokumentů dává za příčinu vzniku nových databázových serverů - s podporou XML a nativních XML databází.

4.2 Typy XML dokumentů

Před samotným prozkoumáním problematiky XML databází a toho, jak se v nich XML soubory ukládají, si rozdělíme XML dokumenty na 2 odlišné typy [8]. První skupinou jsou XML dokumenty zaměřené především na datový obsah. V literatuře jsou nejčastěji označovány jako *datově orientované* (anglicky *data-oriented*, *data-based* nebo *data-centric*). Druhou skupinu tvoří dokumenty, které se vyznačují především smíšeným obsahem ve své struktuře a obvykle jsou určeny pro potřeby člověka než počítače. Nazývají se *dokumentově orientované* (anglicky *document-oriented*, *document-based* či *document-centric*). Oba typy si podrobněji popíšeme v následujících podkapitolách.

4.2.1 Datově orientovaná XML data

Tento typ XML dat se vyznačuje především svou pravidelnou strukturou. Obsah nebývá nijak výrazně smíšený, spíše se opakující. Samotná struktura není pro zpracování až tak důležitá, podstatná jsou obsažená data. Nejčastěji se tyto dokumenty používají jako jakési zapouzdření, do kterého vkládáme data. Nejmenší nezávislé jednotky dat jsou v tomto případě na úrovni prvků typu *atribut* nebo *PCDATA*. Typickým znakem je také fakt, že při práci s datově orientovaným dokumentem nezáleží na pořadí elementů na stejné úrovní (tzv. sourozenců).

Taková data často nejsou tvořena člověkem, nýbrž např. generována nějakou aplikací. Jako příklad si můžeme představit nějaký internetový obchod, kde jsou uchovávány informace o jednotlivých objednávkách jeho zákazníků. Ty jsou poté odesílány jinému systému, který vyřizuje expedici jednotlivých zásilek ze skladu. Taková situace vyžaduje, aby se obě (jak odesílající, tak druhá strana) aplikace spolu domluvily. Snadným řešením může být každou objednávku odeslat jako správně formátovaný XML dokument, jenž tvoří v dnešní době čím dál více oblíbený standard. Druhá strana přesně to očekává a

bude moci data přijmout k dalšímu zpracování. Příklad takové objednávky by mohl vypadat následovně:

Výpis 5: Datově orientovaný XML dokument

```
<objednavka číslo= "GT157T" ze_dne="20.3.2009">
   <dodaci_adresa>
      <jmeno>Pavel</jmeno>
      <prijmeni>Mitko</prijmeni>
      <ulice>1.května 457</ulice>
      <mesto>Zlín</mesto>
      <psc>47557</psc>
   </dodaci_adresa>
   <fakturacni_adresa>
      <jmeno>Pavel</jmeno>
      <prijmeni>Mitko</prijmeni>
      <ulice>1.května 457</ulice>
      <mesto>Zlín</mesto>
      <psc>47557</psc>
   </fakturacni_adresa>
   <zbozi id="12544" nazev="AMD_X2_8400+">
      <kusy>1</kusy>
      <cena>4700</cena>
      <dph_procent>19</dph_procent>
   </zbozi>
   <zbozi id="411" název="GA_MF457−GU">
      <kusy>2</kusy>
      <cena>5100</cena>
      <dph_procent>19</dph_procent>
   </zbozi>
    .....
</objednavka>
```

4.2.2 Dokumentově orientovaná XML data

Dokumenty zastupující tuto skupinu mívají naopak nepravidelnou strukturu, málo se opakující a často i smíšený obsah. Nejmenší jednotkou jsou často elementy obsahující smíšený obsah, ale může to být i celý dokument. Je to právě struktura, která je na celém dokumentu nepostradatelná a jakákoliv záměna pořadí elementů by se projevila na jeho významu. Obvykle je tento typ dokumentu vytvářen člověkem, který do něj zanáší nějakou skutečnost odpovídající realitě (v reálném světě převládá mnohotvárnost) a často je pro člověka i určen. Příkladem takových dokumentů může být kniha, elektronická pošta (email), novinový článek a podobně.

Výpis 6: Dokumentově orientovaný XML dokument

```
<clanek rubrika="sport" datum="10.4.2009">
   <nadpis>Česká osmnáctka napjatě sledovala duel</nadpis>
   <odstavec> Češi po porážkách s Německem 3:4 a favorizovanou
```

```
                Kanadou 3:4 v prodloužení museli napjatě čekat, zda Němci
                neuhrají bod.
    </odstavec>
    <odstavec> Německo bylo v utkání se Švýcary po předchozích
                vystoupeních na turnaji papírovým favoritem. Porazilo Čechy,
                se Švédy bojovalo o vyrovnání do poslední vteřiny.
    </odstavec>
     ......
</clanek>
<clanek rubrika="sport" datum="10.4.2009">
 ......
</clanek>
```

4.3 Databáze s podporou XML

Předchozí rozdělení XML dokumentů na dva typy je důležité z hlediska volby samotného ukládání XML souborů. Při výběru úložiště máme totiž, jak už to bývá, více možností. Je možné použít některou z dostupných relačních databází, která umožňuje uchovávání XML dat. Obvykle jsou takové systémy označovány pojmem *XML-enabled* ("XML podporováno"). Z nejznámějších zástupců můžeme uvést Microsoft SQL Server 2000, Oracle, IBM DB2. I když tyto databázové systémy ukládají svá data ve formě tabulek, dokáží bez problému zpracovat, uložit a vrátit námi požadovaný XML dokument. Avšak pozornost si zaslouží fakt, že databáze může uživateli navrátit XML dokument rozdílně strukturovaný, než jak byl původně uložen. Tento nedostatek odráží způsob, jakým relační databázové systémy mapují XML dokumenty do tabulek. Z toho vyplývá, že tento způsob je dostatečný pro datově orientovaná, ale není příliš vhodný pro dokumentově orientovaná XML data. Právě pro tyto účely byly vyvinuty (a stále se vyvíjejí nové) nativní XML databáze (anglicky *native xml database*).

4.4 Nativní XML databáze

Hlavním úkolem nativní XML databáze je uchování dokumentu přesně tak, jak je tvořen originál. Tzn. včetně jeho kompletní logické struktury a všech prvků, které obsahuje (také komentáře, deklarace atd.). Předpokládá se taktéž, že při požadavku na určitý XML dokument nebo na část nějakého dokumentu (viz další kapitoly), databáze vrátí dokument přesně odpovídající ukládanému originálu. Je jasné, že nativní XML databázové systémy jsou vhodné pro ukládání jak datově tak dokumentově orientovaných XML souborů. To z nich činí nejlepší kandidáty pro volbu úložiště dat tohoto formátu.

Spojení *nativní XML databáze* bylo poprvé použito u projektu Tamino, vedeným firmou Software AG. Dnes je již tento výraz běžně používán všemi vývojáři zabývajícími se touto problematikou. Neustále se tento obrat také vrývá pod kůži širší veřejnosti. V době psaní této práce je již na trhu dostupných, troufám si říci, několik desítek existujících nativních XML databázových systémů. Některé jsou licencovány za nemalé finanční částky a některé se naopak pyšní přívlastkem *open source*, neboli zdarma šiřitelné. Je obtížné bez dalšího zkoumání či testování říci, která z dostupných nativních XML da-

tabází je lepší a která horší. Kritéria, která všechny takové systémy spojují, ať už jsou placené či nikoli, jsou především rychlost zpracování a ukládání jednotlivých XML dokumentů do databáze, podpora celosvětově uznávaných W3C standardů pro následné dotazování nad těmito dokumenty a do třetice opět rychlost, tentokrát provedení dotazu. Nejvýznamnějšími podporovanými standardy pro získávání částí nebo i celých XML dokumentů z databáze jsou jazyky *XPath* a *XQuery*, pro provádění změn v dokumentech pak jazyk *XUpdate*. Jazykům XPath a XQuery jsou věnovány kapitoly 3.2 a 3.3. Přehled několika nejznámějších systémů postavených na nativních XML databázích je uveden v následující tabulce 3. V tabulce je u produktu vždy uvedena aktuální verze, technologie použitá při implementaci, podporované jazyky pro práci s XML daty a tvůrce systému.

Tabulka 3: Příklady nativních XML databázových systémů

Název	**Verze**	**Jazyk**	**Dotazovací jazyky**	**Autor systému**
eXist	1.2	Java	XQuery 1.0, XPath 2.0	Wolfgang Meier - vedoucí projektu
XIndice	1.1	Java	XPath, XUpdate	The Apache Software Foundation
dbXML	2.0	Java	XPath	dbXML Group
Tamino XML server	4.4		XQuery	Software AG
XHive/DB	8	Java	XQuery, XPath, XLink, XPointer, DOM Traversal	X-Hive Corporation
Progress Sonic XML Server	7.6.1	Java	XPath, XQuery	Progress software Corporation
Oracle Berkeley DB XML	2.4.16	C++	XPath, XQuery, XUpdate	Oracle Corporation
MonetDB XQuery	4.28.0	C	XQuery, XUpdate	CWI

Jak je z výčtu na první pohled patrné, naprostá většina nativních XML databází je implementována v prostředí *Javy*. Zcela určitě volí vývojáři tento jazyk právě kvůli jeho snadné přenositelnosti mezi různými platformami (vždyť právě nezávislost na platformě je základním pilířem celého jazyka Java). Společnými rysy všech nativních XML databází jsou především tyto vlastnosti:

- XML data jsou ukládána na úrovni dokumentu (dokument jako nejmenší datový jednotka)
- XML dokumenty obvykle seskupovány do kolekcí
- implementován minimálně jeden dotazovací jazyk, některé disponují více jazyky
- používáno indexování pro zrychlení práce s daty
- řešení transakcí, zamykání a víceuživatelského přístupu

Stejně jako v relačních databázích si můžeme představit jeden řádek libovolné tabulky jako nejmenší jednotku dat, v nativní XML databází je touto jednotkou XML dokument. Jak bylo zmíněno, jednotlivé XML databázové systémy obvykle seskupují XML dokumenty do tzv. kolekcí. Pokud si tedy představíme XML dokumenty jako řádky tabulky, kolekce pak budou logicky odpovídat jednotlivým tabulkám. Některé databáze vyžadují ukládat v rámci jedné kolekce pouze dokumenty sobě podobné. To je zaručeno schématy, kterým svým obsahem dokumenty jedné kolekce musí odpovídat. Taková vlastnost databáze se nazývá závislost na schématu (anglicky *schema-dependent*). Naopak najdeme i XML databáze takové, které žádnou podobnost dokumentů náležících do stejné kolekce nevyžadují. Kolekce pak označujeme jako nezávislé na schématu (*schema-independent*).

Některé, ne však zdaleka všechny, nativní XML databáze umožňují taktéž editaci XML dat. Příkladem z tabulky 3 je systém *XIndice* od firmy Apache Software. K úpravě XML dat je zde používán jazyk *XUpdate*. Úprava XML dokumentů je však nepřímo možná i u jiných systémů. Například tak, že je celý dokument z databáze nahrán, pozměněn (nějakými nástroji pro editaci XML dokumentu), poté opět celý uložen do databáze a původní originál smazán. Princip je vlastně stejný, pouze při použití jazyka pro úpravu XML dat za nás veškeré úkony spojené s editací dokumentu provádí přímo databázový systém.

Indexování uložených dat obecně napomáhá urychlení vyhledávání v těchto datech, tudíž i celkové práci s nimi. Nejinak je tomu u ukládání dat ve formátu XML. Základním typem je indexování všech elementů a atributů v dokumentech. To samozřejmě urychluje zpracování dotazů, při kterých se v datech vyhledává právě podle názvů elementů či atributů. Dotazy tohoto typu jsou ostatně nejčastější. Pokročilejším typem indexování je tzv. full-textové, kdy se indexuje veškerý text obsažený v elementech nebo v hodnotách atributů. Tento typ indexování nemusí být součástí každé nativní XML databáze.

Transakce je další pojem, který nesmí být cizí žádnému modernímu databázovému systému. Stejně je tomu i u systémů určeným pro ukládání a práci s XML daty. Zamykání je možné na úrovní celých XML dokumentů, proto se může stát a stává se, že v jeden moment k určitému XML dokumentu může přistupovat (ve smyslu upravování dokumentu) pouze jeden uživatel. Ostatní uživatelé musí počkat, dokud první uživatel práci s dokumentem neukončí. Na možnosti uzamykání částí dokumentů se již v dnešní době usilovně pracuje a v nejbližší době se pravděpodobně začne v systémech objevovat.

4.5 Popis databáze eXist

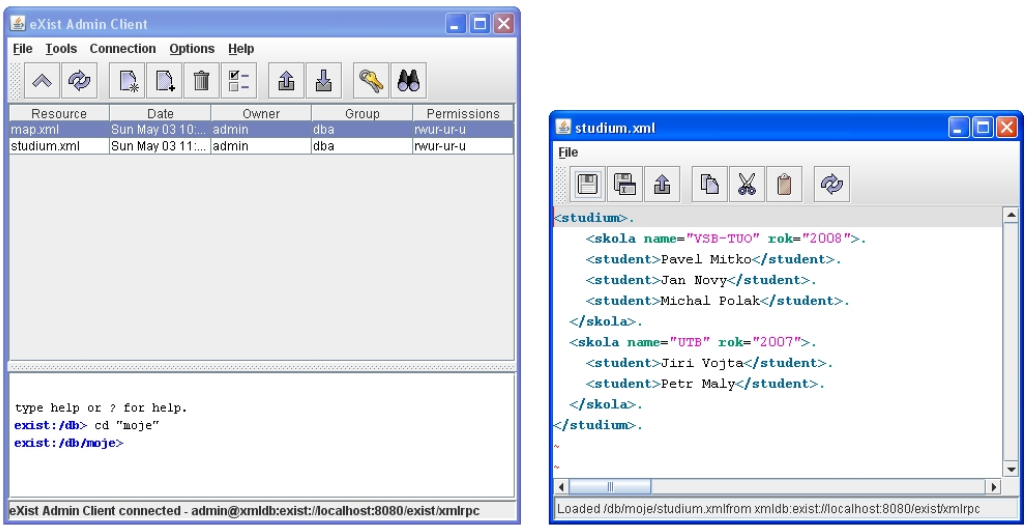
Ze všech mnou prozkoumaných nativních XML databází uvedených v tabulce 3 bylo nutné zvolit jednu, na které poté bude postavena má klientská aplikace. Nakonec jsem se přiklonil k databázi eXist [12]. A to jednak z důvodu volného šíření tohoto produktu (jedná se o open-source databázi) a navíc pro výborně zpracovanou dokumentaci API databáze. API, neboli způsob přístupu do databáze, které jsem pro svou aplikaci zvolil, je blíže popsáno v kapitole 5.2. Celá XML databáze je vytvořena v programovacím jazyce Java a pro její správný běh je nezbytné mít nainstalované JDK (nejlépe aktuální verzi).

Na domovské internetové stránce databáze [12] je vždy k dispozici nejnovější stabilní verze (v době psaní této práce to je soubor **eXist-setup-1.2.5-rev8668.jar**). Po stažení

stačí tento soubor spustit na počítači a započne se samotná instalace. Celý proces instalace je intuitivní. Po uživateli se požaduje pouze zadat adresář pro instalaci a vytvořit si heslo pro pozdější přístup k databáze. To však není povinné, pouze doporučené pro zabezpečení. Doporučuji si také nechat vytvořit ikony na ploše systému, aby byl přístup k databázi co nejjednodušší. Po dokončení instalace stačí spustit databázi ikonou s názvem *eXist Database Startup*.

Systém EXist poskytuje uživateli dvě možnosti přístupu ke své správě. První možností je www rozhraní. Databáze totiž obsahuje integrovaný www server, který se na počítači nastartuje spolu s databází. Implicitně se k této webové stránce přistupuje zadáním adresy **http://localhost:8080/exist/index.xml** do kteréhokoli webového prohlížeče (MS Internet Explorer, Mozilla Firefox, Opera apod.). Na stránce v menu vlevo najdeme možnost přihlášení jako administrátor - uživatelské jméno *admin* (za pomocí hesla, které jsme zadali při instalaci, v opačném případě bez hesla). Poté můžeme provádět základní úkony s databází (přidávat či odebírat XML dokumenty, provádět dotazování, zálohovat celou databázi aj.).

Druhou možností správy databáze je využít předchystaného administrátorského grafického rozhraní. Toto GUI je implementováno taktéž v jazyce Java a spouští se na zmíněné webové stránce odkazem *Launch* v sekci menu Administration. Na úvodní obrazovce je uživatel vyzván zadat uživatelské jméno a heslo. Opět jako jméno uvedeme *admin* a heslo to, které jsme zadali při instalaci. Pokud jsme žádné nevytvořili, necháme pole prázdné. Po úspěšném přihlášení již vidíme samotný program pro správu (obrázek 4). V horní části okna je k dispozici uživatelské menu, pod ním několik užitečných ikok (založení kolekce, vložení XML dokumentu, provádění dotazů atd.). Hlavní část okna tvoří výpis obsahu databáze.



Obrázek 4: Java Admin rozhraní databáze eXist

Osobně dávám přednost používání Java Admin rozhraní pro správu databáze před webovým rozhraním. Pomocí něho jsem se k databázovému systému přihlásil jako administrátor a poté již nebyl problém v databázi vytvořit kolekci s názvem *moje* a vložit do ní dva existující XML dokumenty. Jedním z nich je dokument *studium.xml*, jehož obsah je zobrazen ve výpisu 1 a na jenž se odkazuji v průběhu celé této práce. Druhým je dokument *map.xml*.

Jak již bylo řečeno, jednotlivé dokumenty bývají v XML databázích často hierarchicky uspořádány do kolekcí (můžeme si představit jako složky souborů na počítači). Nejinak je tomu u systému eXist. V hlavní části okna programu se uživateli vypisují jak názvy kolekcí, tak XML dokumentů. Poznat rozdíl je velice jednoduché. XML dokument narozdíl od kolekce obsahuje v názvu příponu *.xml*. Pokud v okně dvakrát klikneme na název kolekce, logicky do ní vstoupíme a program vypisuje její obsah. Pokud však klikneme na XML dokument, otevře se nám nové okno se samotným obsahem dokumentu.

Na obrázku 4 lze vidět výpis obsahu kolekce s názvem *moje* (obsahuje dva xml soubory) a otevřený výpis obsahu dokumentu *studium.xml*. Databáze je tedy plně funkční a obsahuje už i data, nad kterými bude později možné pomocí vlastní klientské aplikace provádět dotazy a vyhodnocovat je.

5 Popis aplikace

Tato kapitola popisuje klientskou aplikaci v jazyku C#, která byla vytvářena souběžně s touto prací. Má za úkol přiblížit čtenáři jednotlivé etapy vývoje, popsat přístup k API vybrané nativní XML databáze a následné připojení k němu. Dále popisuje, jak aplikace funguje a jak získaná data z databáze zobrazuje. V poslední části kapitoly jsou uvedeny výsledky testování vlastních experimentálních metod zobrazovaní či komprese dat ve srovnaní s již existujícími způsoby.

5.1 Návrh implementace

Stěžejním úkolem této bakalářské práce je vytvoření klientské aplikace schopné připojení k nativní XML databázi, odeslání uživatelem definovaného dotazu a následného obdržení výsledku, který poté aplikace uživateli zobrazí odpovídajícím způsobem. Zároveň však, jako dílčí úkoly, mi byla zadána implementace vlastní metody zobrazování výsledného XML souboru a metody komprese XML dat při přenosu přes počítačovou síť.

K celému problému je nutné přistupovat ze dvou pohledů. Za předpokladu, že se ve finální verzi aplikace bude připojovat k existující databázi, zvolené již v kapitole 4.5, je jasné, že v takové situaci nebude možné používat žádné metody komprese při přenosu dat. Nativní XML databáze, které jsem zkoumal, jednoduše takové prostředky neposkytují. Nejinak je tomu u databázového systému eXist. S metodou zobrazení výsledných dat by takový problém nastat neměl, jelikož je už záležitostí čistě klienta, jak bude s přijatými daty nakládat, resp. jak je bude uživateli zobrazovat. Nabízí se otázka, jak tedy celý vývoj aplikace směřovat? Jediným možným řešením, které mi umožní splnění všech zadaných úkolů, je implementace nějaké jednoduché obdoby XML databáze vedle samotného klienta. Slovo "obdoba" bylo použito záměrně, protože se zajisté nebude jednat o plnohodnotnou databázi srovnatelnou s existujícími XML nativními databázemi. Spíše by se mělo jednat o jakousi serverovou aplikaci, která bude schopna reagovat na požadavky přijaté přes počítačovou síť a odpovídat na ně.

5.1.1 Vlastní XML databáze, server-klient komunikace

V první fázi vývoje bude vhodné začít implementací dvou jednoduchých aplikací - serverové a klientské. Serverová část zde bude plnit úlohu již zmíněné vlastní XML databáze. Takto vytvořená databáze nemusí využívat dotazovacích jazyků. Bude dostatečné, když bude schopna na požadavek ze strany klienta odeslat konkrétní XML dokument, který ve svém úložišti obsahuje. Aby bylo prostředí pro komunikaci klienta se serverem vůbec možné, je důležité nejprve zvolit nějaký komunikační protokol, popř. navrhnout svůj vlastní. A to z toho důvodu, že komunikace bude probíhat přes počítačovou síť (lokální, i přes internet). Výjimkou je samozřejmě stav, kdy obě aplikace, jak klient tak server, běží na stejném počítač. Ne vždy tomu ale tak je. Navíc, i v této situaci je potřeba použití nějakého společného komunikačního protokolu. Definice protokolu (převzato z [17]) ve smyslu informatiky zní takto: Protokol je v informatice konvence nebo standard, podle kterého probíhá elektronická komunikace a přenos dat mezi dvěma koncovými body (re-

alizované nejčastěji počítači). V nejjednodušší podobě protokol definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace. Protokoly mohou být realizovány hardwarově, softwarově a nebo kombinací obou.

Jinými slovy jde o společnou "řeč" obou stran. Strana klienta musí vědět, jak přesně požadavek formulovat a v jaké formě očekávat odpověď. Naopak strana serveru nutně musí znát strukturu příchozího požadavku a znát postup při posílání odpovědi zpět klientovi. V mém případě bude nutné zavést množinu zpráv, které se budou posílat od klienta na server a naopak. Zprávy ve smyslu např. "Jsem klient a teď posílám požadavek XXX". Naopak server bude odpovídat např. "Jsem server, odpověď na požadavek XXX je YYY". Takto si komunikaci samozřejmě jen představujeme, reálná implementace může vypadat jinak. Mnoho různých standardů komunikace již existuje, v mém případě bude zřejmě nejlepším řešením návrh vlastního modelu, který budu využívat.

5.1.2 Použití komprese při přenosu XML dat

Jakmile bude splněna první fáze implementace - server bude schopen odeslat klientovi XML dokument a ten jej ve stejné formě obdrží, nic nebude bránit výtvoření vlastní metody komprese při přenosu dat. Server ještě před odesláním dat XML dokument nějakým způsobem zakomprimuje tak, aby byl co nejmenší, a teprve poté jej odešle. Na druhé straně klient musí být nutně schopen obdržený komprimát opět dekomprimovat. Podoba výsledného XML dokumentu u klienta musí být shodná s podobou originálního dokumentu na straně serveru. Z toho plyne, že server musí implementovat metodu komprese a klient naopak metodu dekomprese.

Je logické, že vývoj bude podle následující koncepce:

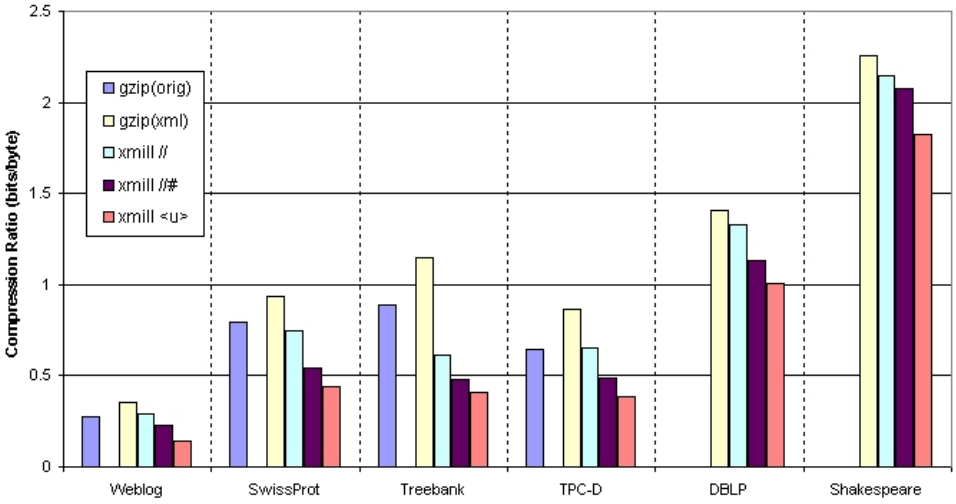
1. Implementovat metodu komprese a zakomponovat ji do serverové části aplikace.
2. Zvrátit proces komprese, čímž vznikne metoda dekomprese, zakomponovat ji do klientské části aplikace.

Při kompresích se obecně snažíme využít toho, že se nějaké části textů (souborů) opakují. Již z povahy struktury XML dokumentů vyplývá, že se v nich jistě budou opakovat názvy elementů i atributů. V praxi tak tomu opravdu je, především u datově orientovaných dokumentů, nejen však u nich. Jeden tentýž element se často v rámci dokumentu opakuje mnohokrát. O co se budeme snažit, je nahrazení všech názvů elementů a atributů za názvy jiné, co možná nejkratší. Přitom si vždy někam poznamenáme, jaké řetězce jsme vyměnili za jaké, aby bylo možné celý proces obrátit. Princip komprese tedy spočívá ve slovníkovém nahrazovaní jmen elementů a atributů v XML dokumentech.

Jako první bylo navrženo nahrazování názvů za postupně narůstající celočíselné hodnoty. Syntaxe jazyka XML však standardně neumožňuje, aby jména elementů začínala číslicí. K tomuto faktu zřejmě budume muset přihlédnout a hledat jiné řešení, např. řetězcové hodnoty (obsahující písmena), které již jsou v syntaxi povoleny.

Jelikož se bude jednat čistě o experimentální metodu komprimace, je nutné ji po dokončení otestovat. Při tomto testování mi bylo doporučeno výsledky srovnat s existujícím

nástrojem pro komprimaci XML souborů s názvem **XMill** [6]. Tento projekt, který byl vyvinut firmou AT&T Labs Research v americkém městě New Jersey v roce 1999, je zdarma k dispozici na internetových stránkách *http://www.liefke.com/hartmut/xmill/xmill.html*. Jedná se o jeden z nejefektivnějších algoritmů komprese pro XML data, ba dokonce možná i nejvýkonější. Je implementován v jazyce C++ a využívá především seskupování řetězců v souborech XML s určitými podobnostmi. Dosahuje lepších výsledků než renomované kompresní nástroje, jako např. *WinZip*, *WinRar* i *gzip*. Pro představu výkonosti XMill komprese je na obrázku 5 srovnání s programem gzip. Obrázek obsahuje výsledky několika způsobů komprese XMill a dvou způsobů komprese gzip.



Obrázek 5: Srovnání výkonu komprese XMill a gzip, podle [6]

Nepředpokládám, že bych svojí metodou komprese překonal nástroj XMill, nicméně to do vývoje přidává jistou výzvu. Grafické porovnání výsledků dosažených vlastní metodou komprese s výsledky komprese XMill jsou uvedeny v kapitole 5.4.1.

5.1.3 Zobrazení XML dat

Dalším dílčím úkolem mé práce je implementace komponenty pro zobrazení XML dokumentu a její následné zapojení do grafického rozhraní klienta. Jako jakási předloha při tvorbě může být použita komponenta s názvem *WebBrowser*, jenž je v implementačním prostředí .NET k dispozici. Jedná se o obdobu programu MS Internet Explorer. Na obrázku 6 je podoba XML dokumentu *studium.xml* (výpis 1) tak, jak ji zobrazí WebBrowser.

Na způsobu zobrazení XML dokumentu pomocí komponenty WebBrowser jsou důležité především dvě skutečnosti.

Zaprvé WebBrowser zvýrazňuje syntaxi jazyka XML použitím barev a stylu písma (tučné). Vidíme, že úvodní deklarace je obarvena modrou barvou. Uzavírací značky elementů (<, > a />) spolu se znaky = a " jsou taktéž zobrazené v modré barvě. Názvy

Obrázek 6: Zobrazení XML dokumentu pomocí WebBrowser komponenty

elementů a atributů jsou tmavě červené a nakonec textové hodnoty elementů i atributů jsou vypsány tučným černým písmem.

Zadruhé u elementů, které obsahují nějaké vnořené prvky, se vlevo zobrazuje malý symbol -. Pokud na tento symbol klikneme, dojde k zabalení elementu, tzn. jeho vnořené elementy (jeho potomci) jsou skryti. Symbol - se poté automaticky změní na +, kterým lze element opět rozbalit.

WebBrowser na první pohled funguje velmi slibně. Nicméně po hlubším prozkoumání bylo zjištěno, že při zobrazování velkých XML dokomentů (1 Mb a více) doba zobrazení rapidně narůstá. Při implementaci vlastní komponenty se pokusíme tomuto problému předejít.

Máme za úkol implementovat komponentu, která by měla obsahovat stejné funkce jako WebBrowser, tzn. jak zvýraznění syntaxe XML jazyka, tak možnost zabalování a rozbalování elementů ve stromové struktuře. Vlastností navíc by měla být také schopnost vyhledávání textu v dokumentu zobrazeném touto komponentou. Pokud je text nalezen, komponenta jej zvýrazní, např. podbarvením textu.

Z vlastních zkušeností již vím, že pokud chceme v prostředí .NET používat barevné výpisy textu, slouží nám k tomu existující kompenenta *RichTextBox*. Jedná se o nadstavbu klasické komponenty *TextBox*, která je velmi hojně používaná i v jiných implementačních prostředích. Předpokládejme tedy, že architektura mé zobrazovací komponenty bude následující:

- bude dědit vlastnosti existující komponenty RichTextBox (schopnost zobrazovat barevný text)

- navíc bude implementovat možnost zabalování a rozbalování stromové struktury

Několikrát již byl zmíněn pojem *RichTextBox*. Jaký je však rozdíl mezi komponentami TextBox a RichTextBox? Základní odlišnost je ta, že TextBox neposkytuje zobrazování barevného textu. Kdežto komponenta RichTextBox, jak už ostatně nese ve svém názvu, umožňuje přiřazení speciálně formátovaného textu, tzv. *RichText Formátu*.

RTF [16] je firmou Microsoft vyvinutý formát textu, který obsahuje bohatou množinu formátovacích příkazů, který je nezávislý na platformě. Vznikl již v roce 1987. Pomocí tohoto formátu je obecně možné vyměňovat dokumenty mezi nejrůznějšími programy pro zpracování textu se zachováním vzhledu a formátu. Na rozdíl od většiny vlastních formátů souborů textových editorů je RTF čitelný i v prosté textové podobě, tedy jeho obsah vypadá jako zvláštní text ASCII, nikoliv jako změť nesmyslných znaků.

Pomocí správně definovaného řetězce textu ve formátu RTF tedy budeme schopni komponentě *RichTextBox* říci, kdy má jakou barvou který text vypsat.

Co se týče implementace funkce zabalování a rozbalování elementů ve vypsaném XML dokumentu, je nutné chápat zobrazený dokument spíše jako jednotlivé řádky textu. Samotné znaky - a + bude rozumnější vykreslovat nalevo od vypsaného textu (mimo plochu s vlastním textem), než přímo do textu, jak je tomu u WebBrowseru. Problém bude pravděpodobně rozložen do tří hlavních úkolů:

1. Určit, ke kterým řádkům (řádky, které obsahují otevírací tagy elementů) znak - nebo + vykreslit.

2. Správně (ve smyslu horizontální pozice) přiřadit vykreslené symboly - nebo + k řádkům textu a zajistit jejich odpovídající posun zároveň s posunem textu.

3. Po zabalení elementu zobrazovaný text překreslit tak, aby vnořené prvky elementu nebyly vidět a naopak.

Podle prvotní analýzy bude zřejmě nutné celý XML dokument, který má být komponentou zobrazen, rozdělit na jednotlivé řádky a s těmi poté pracovat. Pokud bychom totiž toto neprovedli, museli bychom pravděpodobně při manipulaci s kterýmkoli řádkem (při operaci zabalování a rozbalování, tzn. skrývání a odkrývání) vždy procházet celý text od začátku až po námi požadovaný řádek. Struktura a způsob uchovávání jednotlivých řádků textu v paměti bude vyřešena až v rámci implementace.

5.2 Zvolené rozhraní k databázi eXist

Nativní XML databázový systém eXist nabízí programátorům celou řadu rozhraní API, pomocí kterých lze k systému přistupovat (XML:DB API, XML-RPC, REST, SOAP). Všechna jsou velmi dobře zdokumentována v programátorské příručce [13]. My k databázi potřebujeme přistupovat způsobem, který bude nezávislý na tom, zda je klient spuštěn na stejném počítači jako databáze, anebo komunikuje ze vzdálené stanice přes počítačovou síť. Tento a také fakt, že aplikace bude implementována v odlišném programovacím jazyce než nativní XML databáze, nás vede k potřebě nějakého univerzálního rozhraní, které je mimo

jiné dobře použitelné v prostředí internetu. Přesně takovým požadavkům odpovídá rozhraní *REST API* používající *HTTP* protokol k přenosu dotazů a výsledků.

Representational State Transfer (REST) [15] je softwarovou architekturou, která je primárně určena pro realizaci hypermediálních distribuovaných systémů, jakým je i *World Wide Web* (WWW). Pojem "REST" byl poprvé představen v roce 2000 v disertační práci Roye Fieldinga. REST se ve své definici striktně odvolává na množinu základních síťových principů, které určují jak jsou definovány a adresovány registrované síťové zdroje. Výraz REST je často používán pro popis jednoduchého rozhraní, které přenáší specifická data prostřednictvím protokolu HTTP bez dalších dodatečných protokolů jako je SOAP a jemu podobné. Více informací k REST lze nalézt např. v [18].

Použití REST API skrze protokol HTTP je popsáno v samostatné sekci programátorské dokumentace k eXist databázi [14].

HTTP přístup do databáze je umožněn díky integrovanému webovému serveru (více informací v kapitole 4.5). V API jsou definovány čtyři typy HTTP zpráv (GET, POST, PUT a DELETE) a jejich struktury, které je nutné dodržet pro správnou kumunikaci. Implicitně je server nastaven tak, že naslouchá příchozím požadavkům na adrese **http://localhost:8080/exist/rest/**. Server každý požadavek aplikuje na databázi a vrací odpověď s nalezenými daty. Relativní cesty v požadavcích jsou brány jako relativní ke kořenové kolekci dokumentů v databázi. Např. pokud zádáme do internetového prohlížeče následující adresu:

http://localhost:8080/exist/rest/db/moje/studium.xml

server obdrží HTTP zprávu typu GET požadující dokument *studium.xml* v kolekci */db/moje*. Pokud je patřičný soubor nalezen, je poslán zpět klientovi, v opačném případě je odeslána HTTP odpověď s chybovým příznakem "404" (tzn. soubor nenalezen).

Pro jednoduché použití REST API jsou všechny základní operace s daty namapovány do HTTP požadavků. Tabulka 4 znázorňuje podporované metody.

Tabulka 4: REST - Mapování HTTP požadavků na databázové operace

Typ HTTP zprávy	**Operace**
GET	Získá požadovaná data z databáze. Je možné doplnit XPath a XQuery dotazy jako volitelné parametry.
PUT	Nahraje data do databáze. Pokud je to potřeba, kolekce jsou automaticky vytvářeny.
DELETE	Odstraní data (dokument nebo kolekci) z databáze.
POST	Umožňuje odeslat komplexní XQuery dotazy, popř. XUpdate dokument určený pro změnu dat v databázi.

Jak tedy k databázi přistoupit a zaslat uživatelem definovaný požadavek? Je to velmi jednoduché. Jak je uvedeno v tabulce 4, požadavek GET umožňuje přidávání parametrů. Všechny tyto parametry jsou uvedeny v [14]. Jedním z nich je parametr **_query=XPath/XQuery Expression**. *XPath/XQuery Expression* již stačí nahradit dotazem v jazyce XPath nebo XQuery, který si uživatel přeje nad daty před jejich navrácením provést. Provedení XPath dotazu touto metodou nad souborem *studium.xml* by tedy vypadal např. takto:

http://localhost:8080/exist/rest/db/moje/studium.xml?_query=/studium/skola[@name='VSB-TUO']/student

Jednotlivé parametry se v adrese oddělují znakem ?, proto je za názvem souboru *studium.xml* uveden. Na takto odeslaný dotaz nám server po aplikaci na dokument (uvedený ve výpisu 1) odešle následující data:

Výpis 7: Odpověď databáze eXist na XPath dotaz

```
< exist:result  exist:hits ="3"  exist:start ="1"  exist:count="3">
  <student>Pavel Mitko</student>
  <student>Jan Novy</student>
  <student>Michal Polak</student>
</ exist:result >
```

Data jsou správná, resp. odpovídají našemu dorazu (studenti ze školy s názvem VSB-TUO). Celá odpověď je vnořena do elementu *exist:result*. Díky tomu tvoří i přijatá data správně strukturovaný XML dokument. Tento element obsahuje také atribut *hits* určující počet nalezených výskytů. Atributy *start* a *count* sdělují, kolik výsledků je vypsáno, a od kterého výskytu se data vypisují (tyto údaje lze při dotazování měnit dalšími parametry v řetězci požadavku).

Toto rozhraní se jeví jako velmi dobře pochopitelné a jednoduše použitelné. Pro samotnou realizaci spojení klientské aplikace s běžící databází eXist bude tudíž nutné pouze správně strukturovat HTTP požadavek, zakomponovat do něj uživatelem definovaný dotaz v jazyku XPath nebo XQuery a ten poté odeslat na patřičné umístění serveru v síti. Umístěním se rozumí IP adresa serveru (pokud běží klient i server na jednom PC, uvádí se localhost nebo 127.0.0.1) a port protokolu TCP (eXist standardně používá port č. 8080).

5.3 Implementace

Po dokončení kompletního návrhu implementace (kapitola 5.1) začal samotný vývoj aplikace. Jak jsem již uvedl, veškerá implementace je provedena v programovacím jazyce C# za použití architektury .NET. Jednotlivé kroky implementace víceméně korespondují s kroky návrhu. Vývoj tedy začal implementací zmíněné serverové aplikace (dále jen server). Ta bude sloužit jako vlastní XML databáze, především pro prvotní testování přenosu XML dat přes počítačovou síť a později pro vývoj vlastní metody komprese a zobrazování XML dat.

5.3.1 Server-klient komunikace

Je logické, že paralelně se serverem byla implementována i část klientská (dále jen klient). Podle kapitoly 5.1.1 bylo nutné nejprve vytvořit komunikační protokol mezi oběma aplikacemi. Já jsem se ve své aplikaci rozhodl použít vlastní návrh komunikace. Především z důvodu potřeby specifických prvků jak ve struktuře dotazu, tak v následných odpovědích. Samotná komunikace probíhá posíláním zpráv ze strany klienta na server a naopak. Pod pojmem zpráva si v oblasti programování představme nějaká data přesně dané

struktury, která jsou posíláná přes existující počítačovou síť. Struktura zpráv použitých v mém komunikačním protokolu je vyobrazena částečným výpisem kódu 8. V aplikaci potřebuji celkem čtyři druhy zpráv. Všechny tyto typy jsou zachyceny ve výčtovém typu enum s názvem *MsgID*.

- **E_REQUEST_FOR_XML** - zpráva zasílaná při požadavku klienta na nějaký XML dokument (směr od klienta na server)
- **E_RESPONSE_FOR_XML** - zpráva zasílaná při odpovědi ze serveru za předpokladu, že server požadovaný XML dokument má a je připraven ho poslat zpět (směr od serveru na klienta)
- **E_TEXT_MSG** - zpráva zasílaná při odpovědi ze serveru za předpokladu, že server požadovaný XML dokument nenalezl nebo došlo k jiné chybě a nemůže tudíž požadavku vyhovět (směr od serveru na klienta)
- **E_RESPONSE_MAP** - zpráva zasílaná při odpovědi ze serveru, používá se pro posílání přepisovacího souboru potřebného pro dekomprimaci LenA (směr od serveru na klienta)

```
public class CDataChunk
{
    public enum MsgID
    {
        E_REQUEST_FOR_XML = 0,
        E_RESPONSE_FOR_XML,
        E_TEXT_MSG,
        E_RESPONSE_MAP,
    };
    public int  ID;
    public int  nDataSize;
    public int  nCompressed;
    public string  sText;

    public void send(BinaryWriter binaryWriter)
    {  ....  }

    public void recv(BinaryReader binaryReader)
    {  ....  }

    public CDataChunk() { }
```

Výpis 8: Zprávy pro komunikaci server-klient

Zpráva obsahuje celkem čtyři proměnné, z nichž tři jsou celočíselné a jedna má hodnotu řetězce. Při každém posílání zprávy se vytváří nová instance této třídy a všem proměnným jsou přiřazeny patřičné hodnoty. Do proměnné *ID* je vždy uložena hodnota odpovídající typu zprávy (číslo 0,1,2 nebo 3), do proměnné *nDataSize* je přiřazena velikost dat, která budou následovat bezprostředně po této zprávě (např. velikost XML souboru,

který bude následovat po zprávě typu E_RESPONSE_FOR_XML). Dále máme k dispozici celočíselnou proměnnou *nCompressed*, jenž interpretuje typ komprese zvolené klientem. Více o použitých kompresích je uvedeno v kapitole 5.3.3. Poslední proměnná *sText* v sobě obsahuje textový řetězec, který je využíván jak klientem (název požadovaného XML dokumentu), tak serverem. Server, pouze však v případě výskytu chyby při vyřizování požadavku, do této proměnné ukládá textovou zprávu s důvodem chyby (např. "Soubor nenalezen.").

Třída *CDataChunk*, reprezentující komunikaci mezi klientem a serverem, obsahuje také metody *send* a *recv*. Ty slouží pro samotné posílání a přijímání zpráv a ve výpisu 8 jejich obsah není uveden.

5.3.2 Implementace serveru

Po zavedení standardu pro síťovou komunikaci již nic nebrání samotnému vývoji aplikace. Tato kapitola popisuje implementaci serverové části. Jak už bylo nastíněno, bude se jednat o jednoduchou aplikaci, která bude po spuštění naslouchat příchozím požadavkům od klientské aplikace a na ty po zpracování odpovídat jednou z možných typů zpráv uvedených v kapitole 5.3.1.

Celou aplikaci bychom mohli rozdělit na dvě základní části:

- část obsluhující jednotlivá příchozí připojení na uživatelem zvoleném TCP portu
- část vykonávající požadavky

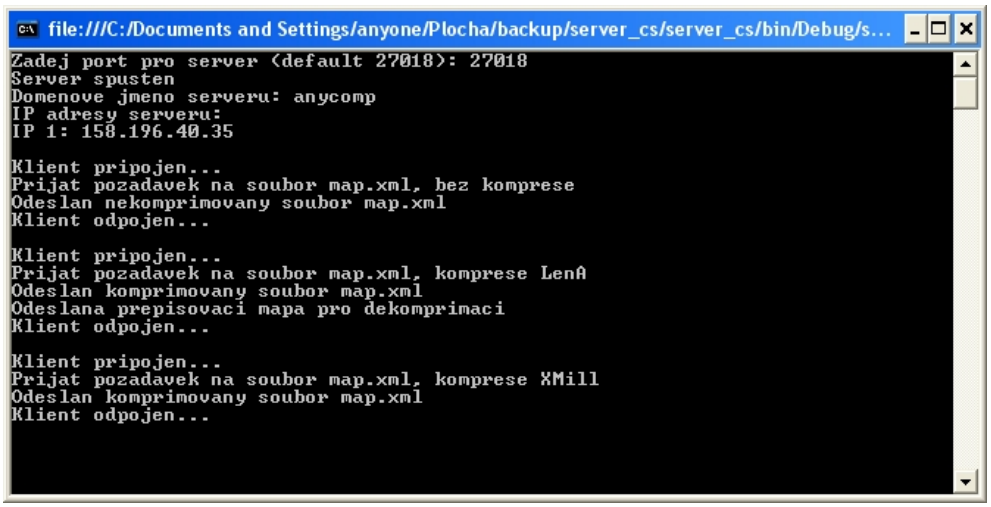
V první části programu bylo vytvořeno rozhraní, které za pomocí tzv. soketů (komponenty umožňující přenost dat mezi vzdálenými počítači) naslouchá síťovému provozu. Je nutné specifikovat číslo portu, na kterém bude server akceptovat příchozí spojení. Tuto hodnotu pak musí znát i klienti, kteří ji pro připojení zadají jako vzdálený port serveru. Číslo portu uživatel zvolí sám (doporučená hodnota od 1024 do 65535) při spuštění serveru, jak lze vidět na obrázku 7. Další důležitou informací pro spuštění serveru je lokální IP adresa stroje, na které server poběží. Tuto hodnotu však není nutné zádavat. Aplikace sama zjistí IP adresu síťového rozhraní počítače, na kterém běží (několik adres v případě více aktivních síťových rozhraní). Kombinací této adresy a zadaného síťového portu bude možné k serveru přistupovat ze strany klientů. V aplikaci bylo také zapotřebí ošetřit situaci, kdy by se k serveru v jeden moment snažilo připojit více klientů najednou. To je vyřešeno pomocí vláken, kdy každého klienta obsluhuje separátní vlákno vytvořené v aplikaci.

Druhá část se skládá ze samotného čtení zpráv s požadavky od klientů a následného sestavení a odeslání odpovědi. Jediný možný požadavek, s kterým v simulovaném prostředí XML databáze budu pracovat, obsahuje název souboru XML, který klient od serveru očekává a způsob, jakým má být XML zkomprimován. S každým takovým požadavkem následuje sled událostí, které musí server provést:

1. Server z požadavku přečte název požadovaného XML souboru a typ požadované komprese.

2. Server se pokusí načíst požadovaný XML soubor. V případě, že soubor nenalezne nebo dojde k chybě při otevírání souboru, odešle klientovi zprávu o chybě a ukončí spojení.
3. Podle požadavku provede nebo neprovede kompresi.
4. Odešle komprimovaný nebo nekomprimovaný XML soubor zpět klientovi. V případě komprese LenA ještě navíc odešle data potřebná pro dekomprimaci.
5. Ukončí spojení s klientem.

Grafické uživatelské rozhraní serverové aplikace není potřeba. Proto byla implementována jako konzolová aplikace. Její běh a jednotlivé výstupy se tudíž zobrazují v klasickém konzolovém okně. Můžeme ho vzhledem přirovnat k příkazovému řádku systému Windows. Ukázka běhu serveru včetně výpisů událostí je na obrázku 7. Na něm je patrné zadání portu, k němuž je uživatel vyzván hned při spuštění. Poté už následuje výpis IP adres lokálního počítače a informace o obsluze jednotlivých požadavků.



Obrázek 7: Serverová aplikace

5.3.3 Použití komprese při přenosu XML dat

Vzhledem k tomu, že princip komprimace byl již uveden v kapitole 5.1.2, v této kapitole je uvedena jen samotná realizace. Při nahrazování názvů elementů a atributů v XML dokumentu jsem se nakonec rozhodl použít znakové řetězce, které při postupném nahrazování narůstají. V anglické abecedě máme k dispozici celkem 26 písmen. Toto číslo ještě zdvojnásobíme, poněvadž můžeme použít malá i velká písmena (v XML se rozlišují malá a velká písmena v názvech tagů i atributů). To nám tedy dává dohromady 52 možností řetězců o jednom znaku. Z praktického hlediska je to velmi výhodné, protože prvních 52 odlišných názvů elementů a atributů v dokumentu můžeme nahradit za jednoznakový

řetězec. A je nadmíru pravděpodobné, že původně všech těchto 52 názvů bude delších než jeden znak. Jakmile se tyto možnosti vyčerpají, jednoduše se délka řetězce zvýši o jeden znak a pokračuje se ve všech kombinacích dvojice znaků.

Tento svůj algoritmus jsem nazval *LenA*[2]. Pro rychlé a snadné pochopení principu nejlépe poslouží krátká ukázka. Jedná se o aplikaci nahrazování na XML dokument ve výpisu 1.

Výpis 9: XML soubor po komprimaci LenA

```
<?xml version="1.0" encoding="utf-8"?>
<A>
  <B C="VSB-TUO" D="2008">
    <E>Pavel Mitko</E>
    <E>Jan Novy</E>
    <E>Michal Polak</E>
  </B>
  <B C="UTB" D="2007">
    <E>Jiri Vojta</E>
    <E>Petr Maly</E>
  </B>
</A>
```

Na výpisu 9 vidíme, že všechny názvy elementů (*studium*, *skola* atd) i atributů (*name*, *rok*) byly nahrazeny za řetězce velkých písmen A až E. K tomuto souboru je ještě nutné vytvořit soubor přepisovacích pravidel (vzniká souběžně s nahrazováním). Tento soubor musí být přiložen k vlastnímu XML dokumentu, aby byla umožněna transformace zpět na původní XML dokument. Přepisovací pravidla pro příklad z výpisu 9 vypadají následovně:

studium * A
skola * B
name * C
rok * D
student * E

Každé pravidlo je tvořeno vždy dvojicí (původní hodnota - nová hodnota), zpravidla oddělenou nějakým speciálním znakem. Zde jsou pro názornost členové dvojice odděleni hvězdičkou (*) a jednotlivé dvojice od sebe odřádkováním. V praxi tomu však může být jakkoli jinak.

Jednotlivé řetězce pro nahrazení jsou v algoritmu tvořeny co možná nejkratší. Jak už bylo řečeno, začíná se jedním znakem a postupně se vyčerpávají další kombinace dvou, tří a více znaků. Intervaly jednotlivých řetězců tak, jak jdou po sobě, bychom mohli vyjádřit takto:

- A..Z (26 možností)

[2]Pojmenování vzniklo podle jedné osoby, pro mě velmi důležité.

- a..z (26 možností)
- AA..Az (52 možností)
- BA..Bz (52 možností)
- CA..zz (2600 možností)

Samozřejmě bychom mohli pokračovat třemi znaky atd., ale to je zbytečné. Ačkoli je tato metoda účinná, poměrně velkou roli hraje různorodost elementů a atributů v dokumentu. To je logické, protože čím více různých názvů elementů a atributů bude nahrazeno, tím více pravidel se objeví v přepisovacím souboru. A ten je nutné připočítávat k velikosti XML dokumentu, jelikož je jeho nedílnou součástí (bez něj bychom původní XML dokument nikdy nezískali zpět). Výsledky komprimace *LenA* ve srovnání s komprimací *XMill* jsou uvedeny v kapitole 5.4.1.

5.3.4 Zobrazení XML dat

Implementace zobrazení XML dat spočívá v programování komponenty navrhnuté v kapitole 5.1.3. Celou tuto komponentu jsem pojmenoval *XMLenAView*[3]. Prvotní strategie implementace byla následující:

1. Vstupem komponenty je XML dokument.
2. XML dokument je následně procházen XML parserem.
3. Na základě parsování je obsah celého dokumentu převeden na jeden dlouhý řetězec textu ve formátu RTF (obsahuje údaje o barvách, stylu písma atd), součástí řetězce jsou i symboly pro "nový řádek".
4. Celý tento RTF řetězec je vložen do komponenty RichTextBox pro vykreslení.

Po celou dobu vývoje komponenty XMLenAView byly její výsledky srovnávány s komponentou WebBrowser. A to především časová náročnost na zobrazení vstupního XML dokumentu komponentou, ale i samotný vzhled zobrazeného textu. Snahou bylo se k WebBrowseru co nejvíce přiblížit. Princip zobrazování uvedený výše se jevil jako dostatečný, zobrazování XML dokumentu funguje jak má. V porovnání s WebBrowserem byly časové rozdíly zanedbatelné. Například doba zpracování XML souboru o velikosti 2 Kb byla: 16 ms u WebBrowser komponenty, 25 ms u komponenty XMLenAView.

Problém ovšem nastal při testování s většími XML dokumenty (v řádu desítek až stovek Kb). Při snaze zobrazit XML dokument o velikosti 500 Kb pracovala komponenta WebBrowser přibližně 20 vteřin, ovšem komponenta XMLenAView něco málo přes 2 minuty. Nárust času je tedy abnormální. Při stejném testu komponenty XMLenAView s XML souborem velkým 2 Mb došlo několikrát i k úplnému zaseknutí programu. Po patřičném prozkoumání jsem došel k závěru, že RichTextBox zřejmě při každé změně

[3]Pojmenování vzniklo podobně jako v případě komprese LenA.

barvy písma znovu vnitřně volá funkci renderování a překresluje veškerý text znovu. To má za následek téměř exponenciální nárust času potřebného k zobrazení dat s rostoucí velikostí XML dokumentu. Bylo tedy nutné vymyslet řešení tohoto problému.

Velmi efektivní se ukázalo využití možnosti přiřazovat do RichTextBoxu v jeden moment jen část textu, resp. tu část, která je viditelná a je nutné ji zobrazit. Vyplývá to ze skutečnosti, že v aplikaci je v okně komponenty vždy vidět jen několik málo řádků (málo ve srovnání s celkovým počtem řádků rozsáhlého dokumentu). Strategie implementace komponenty byla tedy v jistých bodech změněna:

1. Vstupem komponenty je XML dokument.
2. XML dokument je následně procházen XML parserem.
3. Na základě parsování je obsah celého dokumentu reprezentován jako seznam polí (ArrayList), kde jedno každé pole reprezentuje jeden řádek výsledného textu. Text je opět ve formátu RTF.
4. Podle pozice vertikálního posuvníku (snímá posun uživatele v dokumentu) je vypočítáno, které řádky v rámci celého XML dokumentu mají být právě zobrazeny v okně komponenty.
5. Je vytvořen řetězec textu ve formátu RTF obsahující pouze řádky, které je nutné zobrazit (např. 30 řádků z celkových 3000). Tento RTF řetězec je vložen do komponenty RichTextBox pro vykreslení.

Nárust rychlosti je enormní. Vzhledem k náročnosti na vykreslování pouhého zlomku z celkového počtu řádků funguje nyní komponenta mnohem rychleji než WebBrowser. Grafické srovnání časových požadavků na zobrazení různě velkých XML dokumentů je uvedeno v kapitole 5.4.2.

Uložení jednotlivých řádků zobrazovaného textu v paměti počítače (ArrayList) je také výhodné z hlediska implementace následného úkolu, a sice možnosti zabalování a rozbalování stromové struktury zobrazeného XML dokumentu. Ke každému řádku je však potřeba přidat další informace. Jsou jimi údaje, zda u konkrétního řádku má být zobrazena možnost zabalování a v případě, že již takový element byl zabalen, tak je potřebná informace o tom, kolik dalších řádků má být skryto. Z toho důvody byl způsob ukládání jednotlivých řádků později změněn. Již to nejsou pouhá pole řetězců s textem, nýbrž struktury, které kromě samotného textu obsahují ještě zmíněné údaje.

Výsledný vzhled komponenty XMLenAView s implementovaným zvýrazněním syntaxe a s možností zabalování stromové struktury je ukázán na obrázku 8

5.3.5 Připojení k databázi eXist

Připojení k databázové systému eXist bylo realizováno pomocí rozhraní REST API, jehož použití je popsáno v kapitole 5.2. Před odesláním požadavku na databázový server je nutné, aby uživatel zadal tyto údaje:

Obrázek 8: Zobrazení XML dokumentu pomocí XMLenAView komponenty

- IP adresu serveru (dekadický formát, např. 158.196.40.35, popř. localhost)
- TCP port serveru (port, na kterém server naslouchá příchozím spojením)
- XML soubor (soubor v databázi, který požadujeme)
- XPath dotaz (dotaz, který si přejeme nad XML souborem provést)

V případě připojování k systému eXist nemá uživatel k dispozici žádné metody komprese. To je logické, protože eXist žádné komprese při přenosu XML dat neposkytuje. Po zadání výše uvedených údajů může být dotaz odeslán na server. To je implementováno ve čtyřech krocích:

1. Navázat TCP spojení se serverem.
2. Ze zadaných údajů vytvořit správně strukturovaný HTTP dotaz a odeslat jej na server.
3. Přečíst obsah HTTP odpovědi přijaté od serveru.
4. Obsah odpovědi předat komponentám k zobrazení (XMLenAView a WebBrowser, které jsou v klientovi zakomponovány).

Odpověď je uživateli zobrazená jako validní XML dokument, což je zaručeno již před odesláním systémem eXist (viz ukázka ve výpisu 7). Aplikace také umožňuje si zvolit komponentu, která výsledek zobrazí (XMLenAView, WebBrowser nebo obojí). Implementace klienta zahrnuje také vyhledávání v zobrazeném XML dokumentu, které je popsané v kapitole 5.3.6. Nakonec má uživatel možnost si výsledný XML soubor uložit do souboru na disk počítače.

5.3.6 Vyhledávání v XML dokumentu

Po přijetí a zobrazení XML dokumentu, a to jak z vlastní XML databáze tak i z databáze eXist, nabízí aplikace uživateli možnost vyhledávání výrazů v tomto dokumentu (pouze však v komponentě XMLenAView). To je realizováno pomocí prvků v grafickém rozhraní klienta. Zde uživatel zadá výraz, který chce v dokumentu najít a poté oblast hledání. Tím je míněno v jakých částech XML dokumentu chceme výraz hledat. Na výběr má uživatel ze čtyř možností (které může libovolně kombinovat):

- názvy elementů
- názvy atributů
- textové hodnoty (hodnoty elementů)
- hodnoty atributů

K dispozici je také možnost částečného vyhledávání. V tom případě se vyhledává jakýkoliv řetězec, který obsahuje podřetězec zadaný uživatelem. Prakticky se pak v dokumentu vyhledává regulární výraz **.*abcd.***, kde *abcd* zastupuje řetězec zadaný uživatelem. Při každém nálezu požadované hodnoty je tato v komponentě XMLenAView obarvena zelenou barvou. Pokud částečné vyhledávání nepoužijeme, nalezený řetězec musí přesně odpovídat hledanému řetězci.

5.3.7 GUI, lokalizace

GUI klientské aplikace, neboli grafické uživatelské rozhraní, je podrobně popsáno v kapitole Uživatelská příručka, která je součástí příloh.

Lokalizace programu byla provedena do dvou jazyků. Výchozí jazyk aplikace při spuštění je čeština, k dispozici je ještě anglický jazyk. Změna jazyka se provádí v uživatelském menu, v sekci *Jazyk*. Pokud uživatel zvolí jiný jazyk než aktuální, celé grafické rozhraní aplikace se okamžitě začne zobrazovat ve vybraném jazyce.

Samotné texty definující jednotlivé lokalizace jsou uloženy v externích souborech (formát .txt). V těchto souborech jsou uloženy všechny potřebné informace pro zavedení jazyku v aplikaci. Soubor obsahující český jazyk má název **cz.txt** a soubor s angličtinou je **eng.txt**. Je nezbytné, aby oba soubory byly přítomny ve stejném adresáři, z kterého je aplikace spouštěna.

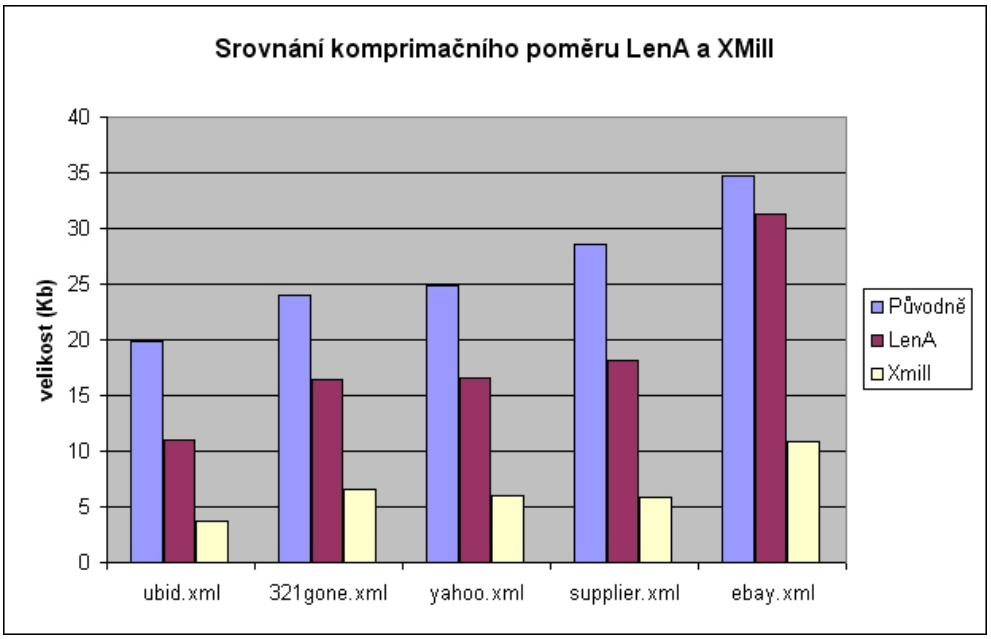
5.4 Výsledky testů

V této kapitole je provedeno několik testů na výkonnost součástí, s kterými jsem v aplikaci pracoval. Jedná se o testy dosažené komprese pomocí vlastního algoritmu LenA a XMill nástroje. Dále o časové srovnání doby komprese obou zmíněných komprimací. A nakonec jsou v kapitole 5.4.2 uvedeny výsledky testu komponent pro zobrazení XML dat (existující WebBrowser a moje vlastní komponenta XMLenAView).

5.4.1 Testy komprese XML dat

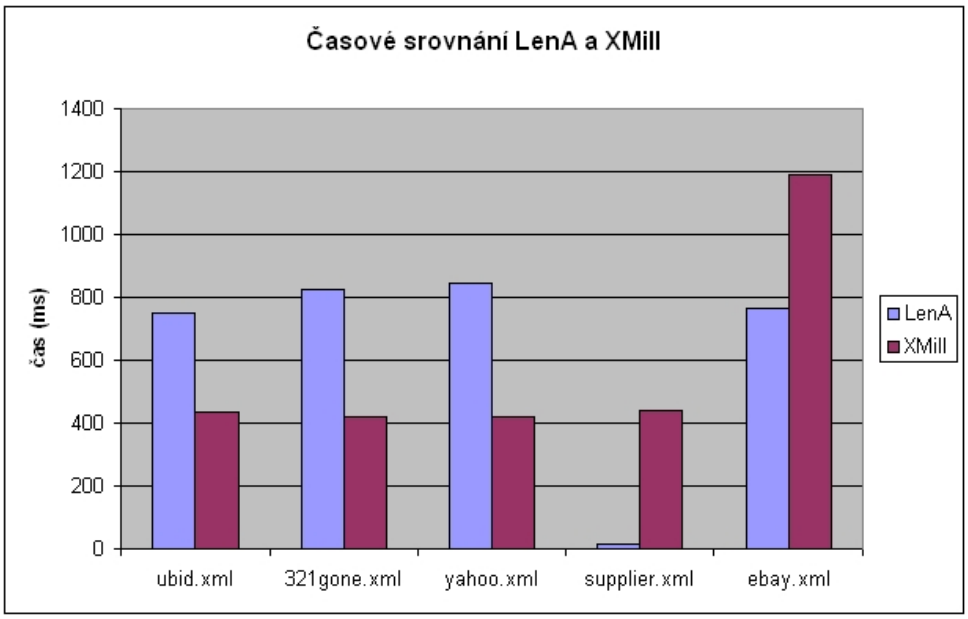
Testy byly provedeny na XML dokumentech, které byly volně staženy na internetu. U každého z pěti testovaných XML dokumentů různých velikostí i struktury jsou v grafech zaneseny časy komprimace, originální velikost souboru a dosažené velikosti po komprimaci LenA a XMill. Grafy veškerých měření jsou na obrázcích 9 a 10. Testované soubory:

1. **ubid.xml** (velikost 19,84 Kb)
2. **321gone.xml** (velikost 23,94 Kb)
3. **yahoo.xml** (velikost 24,83 Kb)
4. **supplier.xml** (velikost 28,56 Kb)
5. **ebay.xml** (velikost 34,73 Kb)



Obrázek 9: Srovnání úrovně komprese LenA a XMill

Na grafu 9 jsou uvedeny hodnoty velikostí souborů v Kb. Jedná se o hodnoty původní před kompresí a výsledné hodnoty po jednotlivých kompresích. Z grafu je patrné, že mnou vytvořená komprese *LenA* není tak výkonná jako použitá komprese XMill. Je tomu tak z toho důvodu, že já ve své kompresi nepoužívám žádné sofistikované algoritmy pro hledání podobných řetězců atd. Pouze nahrazuji XML tagy a atributy za kratší řetězce. Ani v jednom případě nemá LenA lepší výsledek, co se týka velikosti výsledného komprimátu, a zřejmě ani nikdy mít nebude.



Obrázek 10: Srovnání časové náročnosti komprimace LenA a XMill

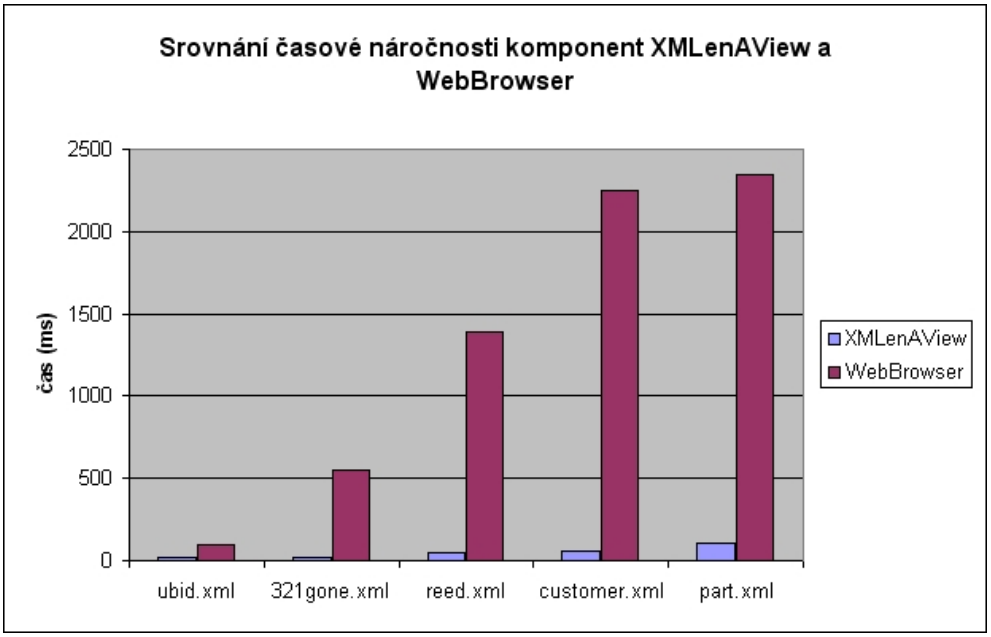
Nyní si popíšeme graf 10. Zde je znázorněn časový průběh jednotlivých operací komprimace. U větších souborů trvá komprimace XMill obvykle delší dobu než LenA. U souborů, jejichž velikost je řádově v jednotkách kB, je rychlost LenA také vyšší než XMill. V nízké časové náročnosti komprimace a dekomprimace (ta časově téměř vždy odpovídá i době dekomprimace) vidím přínos mnou vytvořené komprese LenA.

5.4.2 Testy zobrazování XML dat

Testy zobrazování XML dokumentů spočívají v použití dvou různých komponent pro zobrazení (WebBrowser a má komponenta XMLenAView) a následném srovnání časových údajů. Testované soubory:

1. **ubid.xml** (velikost 19,84 Kb)
2. **321gone.xml** (velikost 23,94 Kb)
3. **reed.xml** (velikost 277,00 Kb)
4. **customer.xml** (velikost 503,57 Kb)
5. **part.xml** (velikost 603,69 Kb)

Na grafu 11 lze vidět časové hodnoty (jednotka milisekunda, tj. tísicina sekundy), za které byly obě komponenty schopny zobrazit vstupní XML dokument. Vlastní komponenta XMLenAView má bezkonkurenčně mnohem lepší výsledky (sloupec v grafu



Obrázek 11: Srovnání časové náročnosti komponent XMLenAView a WebBrowser

je mnohokrát nižší) než WebBrowser nabízený standardně v implementačním prostředí .NET.

6 Závěr

V této práci jsme se po teoretické stránce zabývali především technologií XML, jejím významem v dnešní době a nejčastějším použitím. V kapitole Dotazovací jazyky pro XML byli uvedeni nejznámější zástupci jazyků určených pro vyhledávání v XML datech. Jazykům XPath a XQuery byly věnovány zvláštní kapitoly, jelikož oba tyto jazyky je důležité znát před jakýmkoli dotazovaním nad XML databázemi. Samozřejmě práce není učebnicí těchto standardních pomůcek, spíše jen jakýmsi rychlým průvodcem skrze tyto technologie. Podrobně byla také objasněna problematika nativních XML databází, přičemž jedna z nich byla zvolena pro další plnění zadání této práce.

V kapitole popisující vytvářenou aplikaci byl dodržen doporučený postup při vyvíjení softwaru. Ten se vždy skládá ze specifikace požadavků, návrhu implementace a nakonec samotné implementace. Jelikož v požadavcích stálo i vytvoření vlastních experimentálních metod pro kompresi XML dat a zobrazování XML dat, bylo od začátku jasné, že se mé snahy budou ubírat dvěma paralelními směry.

Jednak bylo nutné vytvoření vlastního prostředí simulujícího XML databázi, které umožnilo plné využití vlastních součástí. V takto vytvořené aplikaci architektury server-klient byla vyvinuta vlastní fungující komprese aplikovatelná na data formátu XML. Tato komprese byla poté testována, porovnávána s již existujícím nástrojem a výsledky zaznamenány do grafické podoby.

Bylo také zapotřebí důkladné prostudování zvolené XML nativní databáze a rozhraní, která poskytuje. Pomocí jednoho z těchto rozhraní byla později vyvíjená klientská aplikace schopna s databází komunikovat a získávat z ní požadovaná data. Napříč oběma hlavními úkoly jsem se potýkal s největší výzvou této práce, a sice implementací vlastní komponenty pro zobrazení XML dat na straně klienta, která by svou funkčností předčila komerční komponentu WebBrowser. S tímto krokem souviselo hned několik problémů, které se postupně při realizaci objevovaly. Především pak rychlost komponenty při zpracování XML dokumentů větších objemů. Nicméně i toto se podařilo vyřešit a výsledky testování byly opět zdokumentovány. Celá tato komponenta je samozřejmě navržená tak, aby bylo kdykoli možné její přenesení do libovolného jiného projektu pracujícím s dokumenty XML. Např. při tvorbě klientů pro jiné nativní XML databáze, než je systém eXist.

7 Reference

[1] TRAVIS, Brian E. *XML a SOAP: Programování serverů BizTalk.* Vydání první. Praha: Computer Press, 2000. ISBN 80-7226-303-X.

[2] SKONNARD, Aaron, GUDGIN, Martin. *XML - pohotová referenční příručka.* První vydání. Praha: Grada Publishing, 2006. ISBN 80-247-0972-4.

[3] W3C Consortium. *XQuery 1.0 and XPath 2.0 Functions and Operators* [online]. c2004, poslední revize 29.10.2004. <http://www.w3.org/TR/2004/WD-xpath-functions-20041029/>

[4] VOCHOZKA, Josef. *Značkovací jazyky a XML (2)* [online]. c2001, poslední revize 14.4.2009. <http://www.ics.muni.cz/zpravodaj/articles/205.html>

[5] W3C Consortium. *XML Schema Part 2: Datatypes* [online]. c2001, poslední revize 2.5.2001. <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

[6] AT&T Labs-Research. *XMill - An Efficient Compressor for XML* [online]. c2004. <http://www.liefke.com/hartmut/xmill/xmill.html>

[7] KOSEK, Jiří. *Kapitola 3. XML schémata* [online]. c2004. <http://www.kosek.cz/xml/schema/wxs.html>

[8] ŽIŽKA, Petr. *Nativní XML databáze - nástin teorie* [online]. c2003. <http://interval.cz/clanky/nativni-xml-databaze-nastin-teorie/>

[9] KRÁTKÝ, Michal. *XML technologie, datový model, API* [online]. c2008, poslední revize 6.4.2009. <http://www.cs.vsb.cz/kratky/courses/2008-09/tis/download/tis-8.pdf>

[10] KRÁTKÝ, Michal. *Nativní XML databáze* [online]. c2008, poslední revize 6.4.2009. <http://www.cs.vsb.cz/kratky/courses/2008-09/tis/download/tis-9a.pdf>

[11] WALSH, Norman. *A Technical Introduction to XML* [online]. c2008. <http://www.xml.com/pub/a/98/10/guide0.html>

[12] eXist. *eXist - Open Source Native XML Database* [online]. c2000. <http://exist.sourceforge.net/>

[13] MEIRE, Wolfgang M. *eXist - Developer's Guide* [online]. c2000. <http://exist.sourceforge.net/devguide.html>

[14] MEIRE, Wolfgang M. *eXist - Developer's Guide: REST-Style Web API* [online]. c2000. <http://exist.sourceforge.net/devguide_rest.html>

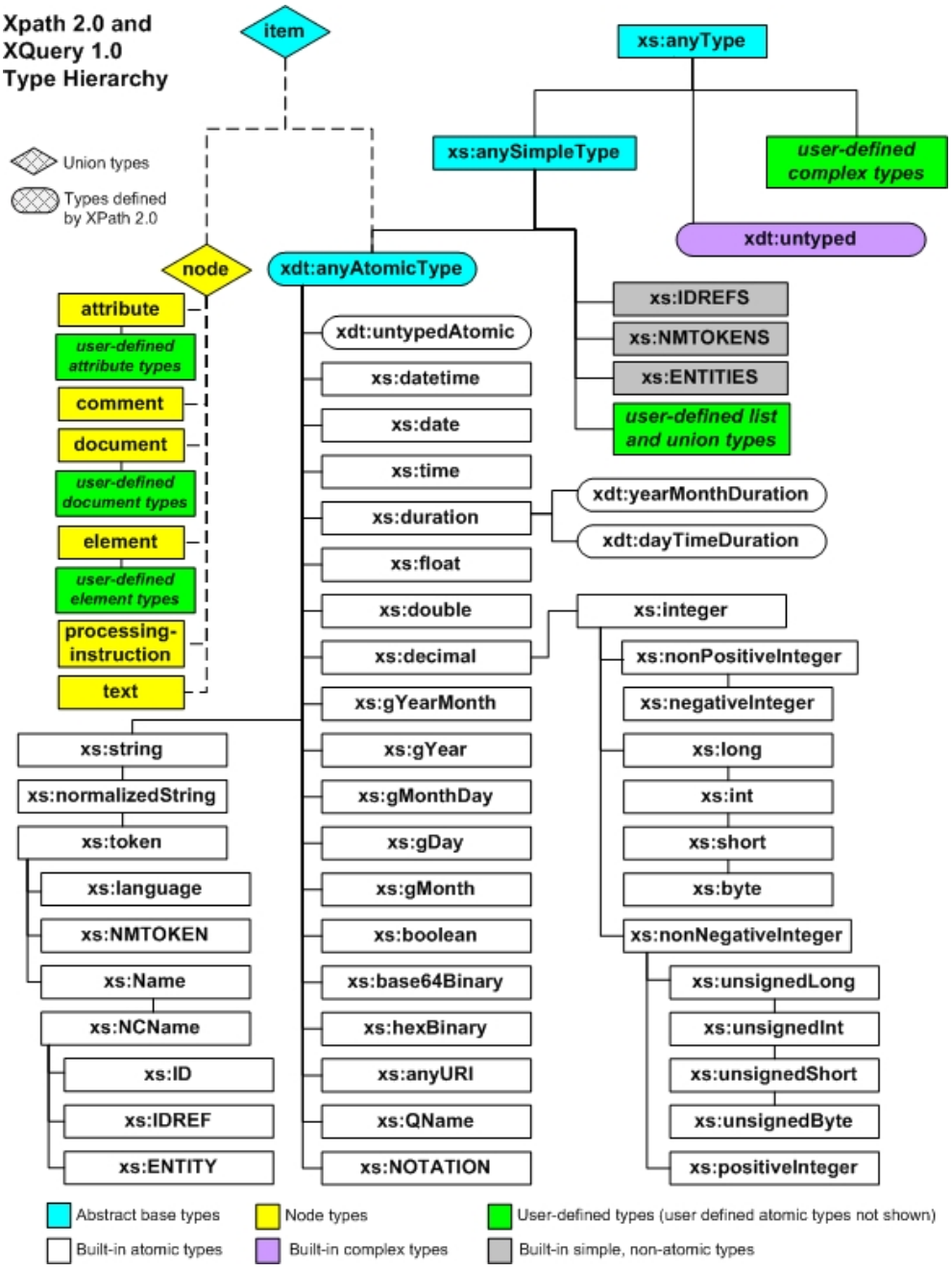
[15] ŠELIGA, Michal. *Orchestrace geowebových služeb* [online]. c2008, poslední revize 12.1.2008. <http://kokos.vsb.cz/wiki/images/e/e5/Seliga1.pdf>

[16] Microsoft Corporation. *Rich Text Format (RTF) Specification, version 1.6* [online]. c2009. <http://msdn.microsoft.com/en-us/library/aa140277.aspx>

[17] Wikipedia. *Wikipedie - Otevřená encyklopedie* [online]. c2002. <http://cs.wikipedia.org/wiki/Wikipedie>

[18] Wikipedia. *Wikipedia - The Free Encyclopedia* [online]. c2001. <http://en.wikipedia.org/wiki/Main_Page>

A Datové typy v jazyce XPath 2.0 a XQuery 1.0



Obrázek 12: Datové typy v jazyce XPath 2.0 a XQuery 1.0, podle [3]

B Uživatelská příručka

Cílem této uživatelské příručky je seznámit uživatele se základy používání aplikace.

B.1 Instalace

Instalace aplikace není nutná. Pro běh programu stačí spustit příslušný soubor podle následující kapitoly - Spuštění aplikace.

B.2 Spuštění aplikace

Spuštění aplikace se provede otevřením souboru **client_cs.exe**. Cesta k tomuto souboru na přiloženém CD je \client_cs\client_cs\bin\Release\. Po spuštění programu se objeví okno s grafickým uživatelským rozhraním. Pro připojení k vlastnímu XML serveru s možností komprese XML dat je nutné nejprve tento server uvést do provozu. To provedeme spuštěním souboru **server_cs.exe**. Umístění souboru na CD je \server_cs\server_cs\bin\Release\. Po spuštění serverové aplikace je uživatel požádán o zadáni TCP portu pro příchozí připojení. Výchozí číslo portu je 27018.

B.3 Grafické uživatelské rozhraní (GUI)

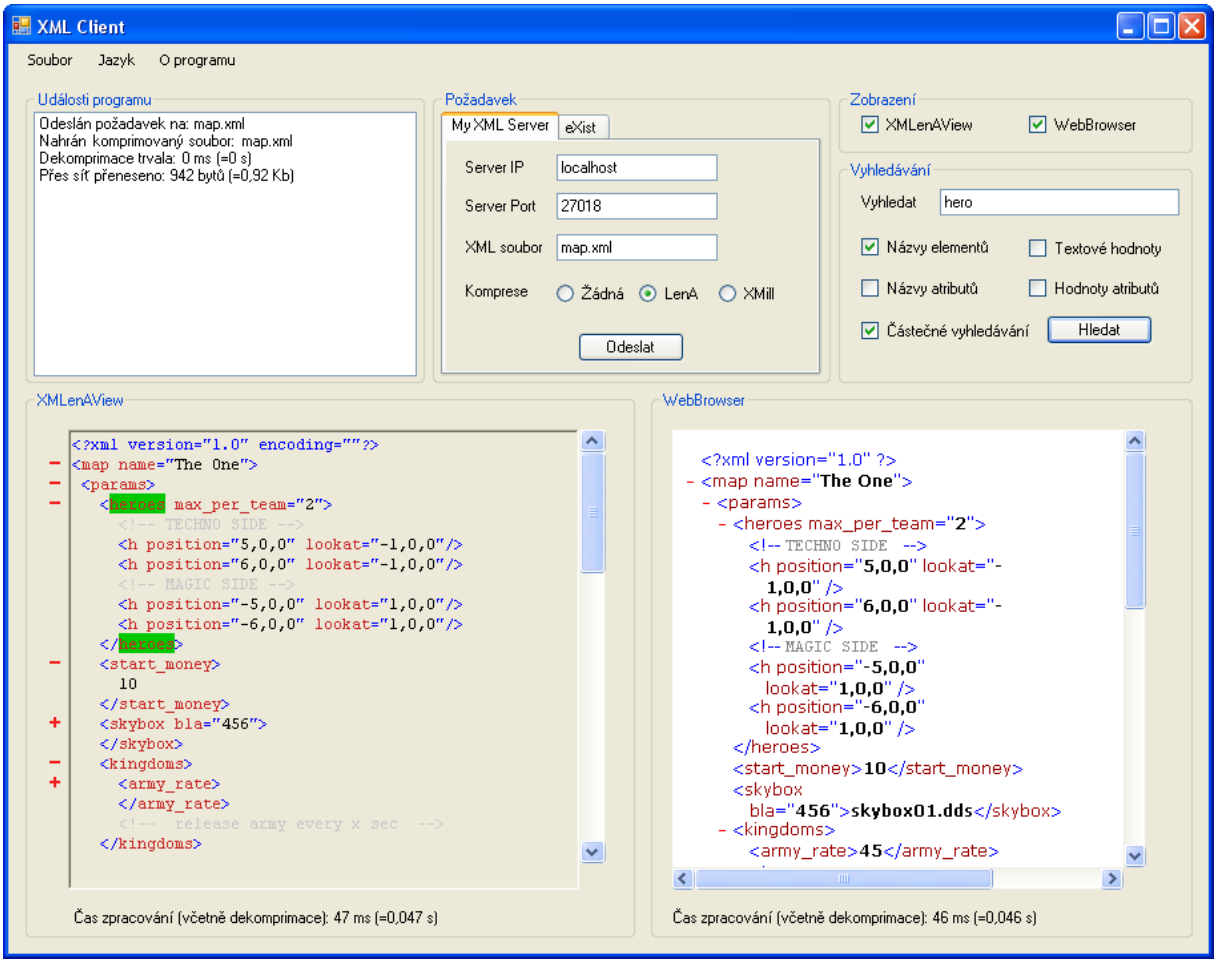
Na obrázku 13 je ukázka okna klientské aplikace tak, jak se uživateli zobrazí po spuštění. Popíšeme si nyní jednotlivé položky v programu a jejich funkce (popsáno bude GUI v českém jazyce).

K dispozici je uživatelské menu, které je v programu umístěno úplně nahoře, pod lištou okna s názvem programu. Položky menu jsou následující:

- Soubor
  - Uložit XML - slouží pro uložení přijatých XML dat do souboru na disku počítače.
  - Ukončit program - uzavře program, neuložená data budou ztracena.
- Jazyk - slouží pro přepínání jazyka aplikace (na výběr je český a anglický jazyk).
- O programu - zobrazí informace o aplikaci (název, autor, verze programu atd.).

Jako první zleva vidíme v okně oblast nazvanou *Události programu*, kde program vypisuje informace o právě probíhajících procesech. Tato část zobrazuje jak zprávy o úspěšně vykonaných příkazech a statistiky prováděných požadavků, tak zprávy chybové (ty jsou vypisovány červenou barvou).

Druhou skupinou v grafickém rozhraní programu je *Požadavek*. Zde jsou ovládací prvky rozděleny do dvou záložek - *My XML Server* a *eXist*. Jednotlivé záložky odpovídají typu XML databáze, ke které se chceme připojit. Při výběru *My XML Server* je aplikací před odesláním požadavku očekáváno vyplnění položek *Server IP*, *Server Port* a *XML soubor*. Poté má uživatel možnost zvolit jednu z implementovaných kompresí pro přenos XML dat.



Obrázek 13: GUI klientské aplikace

Naopak při zvolení záložky *eXist* se uživateli místo komprese nabízí zadání dotazu pomocí jazyka XPath, který bude databází eXist nad požadovaným XML dokumentem proveden. V obou případech se samotný požadavek na server odesílá stisknem tlačítka *Odeslat*.

V třetí části nazvané *Zobrazení* má uživatel možnost volit mezi komponentami XMLenAView a WebBrowser, které se následovně použijí pro zobrazení XML dat přijatých ze strany serveru.

Poslední sekcí určenou k interakci s uživatelem je *Vyhledávání*. Zde se uživateli nabízí hledání jakéhokoli textu v XML dokumentu. Můžeme volit ze čtyř oblastí, v kterých si přejeme text vyhledávat. Před samotným potvrzením vyhledávání tlačítkem *Hledat* ještě můžeme využít možnosti *Částečného vyhledávání*.

Největší část okna tvoří komponenty určené pro zobrazení přijatých XML dat - *XMLenAView* a *WebBrowser*. Pod oběma komponentami se po provedení každého požadavku zobrazí čas, za který komponenta přijatá data zpracovala a zobrazila na obrazovku.

C Obsah přiloženého CD

V kořenovém adresáři přiloženého CD se nacházejí tyto adresáře:

- **bakalářská práce** - obsahuje tuto práci ve formátu pdf a LaTex
- **client_cs** - obsahuje zdrojové soubory klientské aplikace (Klient nativní XML databáze)
- **server_cs** - obsahuje zdrojové soubory serverové aplikace